Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matjaž Rajnar

Primerjava uporabe SOAP in REST za potrebe povezave mobilnih naprav s spletnimi storitvami

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Damjan Vavpotič

Ljubljana, 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V okviru diplomskega dela preučite načine za povezavo mobilnih naprav s spletnimi storitvami, pri tem se osredotočite zlasti na standarda SOAP in REST. Predstavite oba standarda ter ju medsebojno primerjajte po različnih smiselno izbranih kriterijih. Za namen primerjave izdelajte tudi mobilna odjemalca in odgovarjajoča strežnika, en par z uporabo SOAP in drug par z uporabo REST, nato pa primerjajte njuno učinkovitost. Ugotovitve primerjave predstavite in kritično ovrednotite.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matjaž Rajnar, z vpisno številko **63080164**, sem avtor diplomskega dela z naslovom:

Primerjava uporabe SOAP in REST za potrebe povezave mobilnih naprav s spletnimi storitvami

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Damjana Vavpotiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 10. maja 2014 Podpis avtorja:

Zahvaljujem se prof. dr. Damjanu Vavpotiču za mentorstvo ter za vse nasvete in pomoč pri izdelavi diplomskega dela.

Za vzpodbudo in podporo pri študiju se zahvaljujem tudi svojima staršema ter dedku in stricu, ki sta mi v času študija omogočala bivanje v njunem stanovanju.

Zahvala gre tudi sošolcem in prijateljem, brez katerih čas študija ne bi bil niti približno enak.

Kazalo

Seznam uporabljenih kratic

ADT	(*angl.* Android Development Tools) razvojna orodja za Android
API	(*angl.* Application Programming Interface) programski vmesnik
AVD	(*angl.* Android Virtual Device) navidezna naprava Android
CRUD	(*angl.* Create Read Update Delete) ustvarjanje, branje, posodabljanje, brisanje
DDMS	(*angl.* Dalvik Debug Monitor Server) strežnik za razhroščevanje in nadzor Dalvik
GUI	(*angl.* Graphical User Interface) grafični uporabniški vmesnik
HTTP	(*angl.* Hyper Text Transfer Protocol) hipertekstovni prenosni protokol
JAX-RS	(*angl.* The Java API for RESTful Web Services) javanski vmesnik za spletne storitve REST
JAX-WS	(*angl.* The Java API for XML Web Services) javanski vmesnik za spletne storitve XML
JDK	(*angl.* Java Development Kit) razvojno okolje za Javo
JSON	(*angl.* JavaScript Object Notation) oblika za izmenjavo podatkov
JSP	(*angl.* JavaServer Pages) tehnologija za pomoč pri ustvarjanju dinamičnih spletnih strani
JSR	(*angl.* Java Specification Request) dokument, ki opisuje predlagane specifikacije in tehnologije

REST (*angl.* Representational State Transfer) arhitektura za izmenjavo podatkov med spletnimi storitvami

SDK (*angl.* Software Development Kit) razvojno okolje za programsko opremo

SOA (*angl.* Service-oriented architecture) storitveno usmerjena arhitektura

SOAP (*angl.* Simple Object Access Protocol) protokol za izmenjavo podatkov med spletnimi storitvami

UDDI (*angl.* Universal Description Discovery and Integration) mehanizem za registracijo in iskanje spletnih storitev

W3C (*angl.* World Wide Web Consortium) konzorcij svetovnega spleta

WS (*angl.* Web Services) spletne storitve

WSDL (*angl.* Web Service Description Language) dokument namenjen opisovanju razpoložljivih spletnih storitev

WSOAP (*angl.* Wireless SOAP) nabor optimizacijskih tehnik za mobilne naprave

XML (*angl.* Extensible Markup Language) razširljiv označevalni jezik

XML-RPC (*angl.* XML Remote Procedure Call) klic oddaljene procedure, ki temelji na XML

Povzetek

V diplomskem delu obravnavamo uporabo spletnih storitev SOAP in REST na mobilnih napravah Android. Spoznamo, kaj so spletne storitve, kako delujejo in zakaj jih potrebujemo. Predstavimo oba vodilna pristopa, si ogledamo, kako delujeta ter kaj se izmenjuje med odjemalcem in strežnikom. Raziščemo, kako je posamezna tehnologija podprta na mobilnih napravah Android in katere knjižnice so nam pri tem na voljo. S pridobljenim znanjem izdelamo mobilno aplikacijo, ki se poveže na spletno storitev in prenaša datoteke različnih velikosti. To nam omogoča še primerjavo obeh tehnologij s tehničnega vidika, kjer nas zanimajo predvsem končna velikost mobilne aplikacije, količina prenesenih podatkov, število prenesenih paketov, čas prenosa in prepustnost. Rezultati meritev in ugotovitve bodo tako razvijalcem mobilnih aplikacij omogočali lažjo izbiro prave tehnologije.

Ključne besede: spletna storitev, SOAP, REST, mobilna naprava, Android, odjemalec, strežnik, mobilna aplikacija

Abstract

In the thesis we discuss the use of SOAP and REST web services on Android-based mobile devices. We learn, what web services are, how they work and why we need them. We present both leading approaches, look at how they work and what is being exchanged between the client and the server. We study, how each technology is supported on Android-based mobile devices and which libraries are available for use. With this knowledge we develop a mobile application that connects to a web service and transfers files of various sizes. This enables us to further compare both technologies from a technical point of view, where we are mainly interested in the final size of the mobile application, amount of data transferred, number of packets transferred, transfer time and throughput. The results and findings will ease the choice of the right technology for mobile application developers.

Keywords: web service, SOAP, REST, mobile device, Android, client, server, mobile application

Poglavje 1

Uvod

Živimo v obdobju, v katerem mobilne naprave in tablice iz dneva v dan pridobivajo na priljubljenosti. Življenja brez njih si pravzaprav ne znamo več predstavljati. Da bi lahko med sabo povezali naprave, ki temeljijo na različnih operacijskih sistemih in programskih jezikih, so že v preteklosti bile razvite najrazličnejše tehnologije, med katerimi sta bili najbolj prevladujoči SOAP in REST. Omogočale so medsebojno povezovanje osebnih računalnikov in izpostavljanje funkcij naprav v obliki storitev. S prihodom mobilnih naprav in tablic se je pojavila želja, da bi do istih storitev lahko dostopali kar neposredno iz njih. Obstoječe knjižnice za podporo SOAP in REST so bile prevelike in premalo optimizirane, da bi se lahko izvajale na napravah z omejenimi viri, zato so različne skupine začele razvijati prilagoditve omenjenih knjižnic. Cilj diplomske naloge je analizirati oba vodilna pristopa pri razvoju spletnih storitev za mobilne naprave Android iz različnih zornih kotov in prikazati razvoj mobilne aplikacije za preizkušanje obeh pristopov, ki bo teorijo podprla z dokazi.

V poglavju 2 bomo skozi različne definicije spoznali pojem spletnih storitev in si ogledali, kako delujejo. Izpostavili bomo tudi interoperabilnost ali z drugimi besedami neodvisnost od platforme naprave in določenih tehnologij.

V poglavjih 3 in 4 se bomo seznanili s protokolom SOAP in arhitekturo REST. Na kratko si bomo ogledali, kako delujeta, kaj se izmenjuje med

odjemalcem in strežnikom ter izpostavili prednosti in slabosti pri uporabi za oba pristopa. Pri protokolu SOAP bomo na kratko predstavili tudi dokument WSDL in njegovo strukturo.

V poglavju 5 bomo spoznali, kako lahko uporabimo SOAP in REST na Androidu, katere knjižnice so nam na voljo in kako je potekala prilagoditev. Pri protokolu SOAP bomo omenili tudi možnost ročne gradnje sporočil SOAP in druge možnosti za dostop do spletnih storitev, ki ne zahtevajo dodatnih knjižnic, pri arhitekturi REST pa se bomo osredotočili na dobre prakse pri razvoju spletne storitve in odjemalca.

V poglavju 6 bomo na kratko predstavili prednosti uporabe mobilnih aplikacij pred uporabo običajnega spletnega brskalnika.

V poglavju 7 se bomo seznanili z eksperimentalnim in razvojnim okoljem. Ogledali si bomo lastnosti uporabljenih naprav in njihovo vlogo pri preizkušanju.

V poglavju 8 si bomo ogledali implementacijo spletne storitve in odjemalca za oba pristopa. Predstavili bomo funkcije spletne storitve in osnovni princip delovanja mobilne aplikacije.

V poglavju 9 bomo predstavili, kako je potekalo preizkušanje razvite spletne storitve in odjemalca in si ogledali rezultate meritev.

V poglavju 10 bomo zbrali sklepne ugotovitve diplomskega dela, ki bodo razvijalcem mobilnih aplikacij omogočale lažjo izbiro prave tehnologije, in opis dodatnih meritev, ki bi jih bilo zanimivo opraviti v prihodnosti.

Poglavje 2

Spletne storitve

Izraz spletne storitve (*angl.* Web Services) se v današnjem času pogosto uporablja, čeprav nima vedno enakega pomena. Z izrazom spletne storitve najpogosteje označujemo aplikacije, ki so preko omrežja dostopne drugim aplikacijam [1].

Včasih se spletne storitve zamenjuje s spletnimi stranmi. Pomembna razlika med njimi je ta, da spletne storitve nimajo grafičnega uporabniškega vmesnika (*angl.* Graphical User Interface) [2]. Spletna stran je zbirka spletnih dokumentov in povezav med njimi, ki lahko vsebuje več spletnih storitev. Nekatere delujejo samostojno, nekatere pa se medsebojno povezujejo.

Natančnejšo definicijo je podal konzorcij UDDI, ki spletne storitve označuje kot samostojne aplikacije, ki imajo odprte, internetno usmerjene in na standardih temelječe vmesnike [1]. Navedena definicija je bolj podrobna in poudarja potrebo po združljivosti z internetnimi standardi. Zahteva, da je storitev odprta, kar pomeni, da ima objavljen vmesnik, do katerega je možno dostopati preko omrežja.

Konzorcij svetovnega spleta (*angl.* World Wide Web Consortium) je naredil še korak dlje in spletne storitve opredelil kot programsko aplikacijo določeno z identifikatorjem URI, katere vmesnike je mogoče opredeliti, opisati in odkriti kot objekte XML. Spletne storitve podpirajo neposredno interakcijo z drugimi napravami. Uporabljajo sporočila, ki temeljijo na XML,

izmenjava sporočil pa poteka preko standardov, ki temeljijo na internetu [1]. Navedena definicija je precej natančna in predstavi tudi, kako bi spletne storitve morale delovati.

2.1 Dva načina uporabe

Spletno storitev sestavljata odjemalec in strežnik, ki odjemalcem ponuja določene storitve. Prikaz komunikacije med strežnikom spletne storitve in različnimi vrstami odjemalcev je prikazan na Sliki 2.1. Za primer vzemimo aplikacijo za vreme, kjer uporabnik na odjemalcu vnese svoj domači kraj ali poštno številko. Spletni strežnik nato obdela podatke iz zahteve in odjemalcu vrne odgovor z vremensko napovedjo. Odjemalci so lahko različnih vrst; od osebnih računalnikov, pametnih televizorjev in številnih predvajalnikov, do zelo priljubljenih mobilnih naprav in tablic. Glede na način, kako je bila zasnovana spletna storitev, bo na odjemalcih teklo zelo malo ali nič programske kode, kar je še posebej pomembno za mobilne naprave in tablice.

2.1.1 Ponovna uporaba elementov aplikacije

Obstajajo stvari, ki jih aplikacije pogosto potrebujejo. Spletne storitve lahko ponudijo elemente aplikacije, kot so pretvornik valut, podatki o vremenu, prevajalnik besedila in podobno, v obliki storitev [4].

2.1.2 Povezava z obstoječo programsko opremo

Spletne storitve lahko pomagajo pri reševanju težav z interoperabilnostjo, tako da različnim aplikacijam ponudijo možnost za povezavo in tako omogočijo izmenjavo podatkov med njimi [4].

Slika 2.1: Prikaz komunikacije med strežnikom spletne storitve in različnimi vrstami odjemalcev (na podlagi [3]).

2.2 Zahteva po interoperabilnosti

Medtem ko različne platforme lahko dostopajo do spleta z uporabo spletnih brskalnikov, interakcija med njimi ni mogoča brez spletnih aplikacij, ki so namenjene izvajanju na spletu [4]. Zasnovane so na standardih, na katerih temeljijo spletni brskalniki. Z uporabo spletnih storitev lahko te aplikacije svoje funkcije in sporočila izpostavijo navzven, kar omogoča povezovanje med različnimi napravami, neodvisno od platforme naprave in določenih tehnologij. Najbolj razširjena oblika za izmenjavo podatkov med spletnimi storitvami je razširljiv označevalni jezik (*angl.* Extensible Markup Language).

Za potrebe porazdeljenega modela računalništva so bile razvite tehnologije CORBA, ORBacus in Java RMI, vse pa so imele slabo podporo interoperabilnosti. Za reševanje te težave je konzorcij svetovnega spleta razvil protokol SOAP (*angl.* Simple Object Access Protocol), ki uporablja standarda XML in HTTP (*angl.* Hyper Text Transfer Protocol) za lažjo izmenjavo med posameznimi spletnimi aplikacijami. Drug pristop pri razvoju spletnih storitev predstavlja REST (*angl.* Representational State Transfer). REST je množica arhitekturnih načel za načrtovanje spletnih aplikacij, ki se osredotočajo na naslavljanje in prenašanje virov preko HTTP.

Poglavje 3

SOAP

SOAP je na XML temelječ in razširljiv protokol za izmenjavo strukturiranih informacij pri izvajanju spletnih storitev v računalniških omrežjih [5]. SOAP zagotavlja komunikacijo med aplikacijami, ki se izvajajo na različnih operacijskih sistemih, uporabljajo različne tehnologije in programske jezike [6]. Za prenos podatkov se lahko uporablja poljuben prenosni protokol. Običajno je to HTTP, ker ga podpirajo vsi spletni brskalniki in strežniki, zahteve pa niso blokirane s strani požarnih zidov in strežnikov *proxy*. Junija 2003 je konzorcij svetovnega spleta standardiziral SOAP, da bi lahko izpostavili poslovne podatke z uporabo vmesnikov ali storitev, ki izmenjujejo celotne dokumente ali preslikajo podatke na objekte z uporabo metod ter vhodnih in izhodnih parametrov.

3.1 Sporočilo SOAP

Dokument XML postane sporočilo SOAP, ko:

- ogradimo XML s telesom SOAP,
- telo SOAP ovijemo z ovojnico SOAP,
- neobvezno vključimo zaglavje SOAP,
- določimo imenska področja,

- upoštevamo pravila za serializacijo podatkovnih tipov, ki jih določa SOAP,
- vse skupaj povežemo s prenosnim protokolom.

```
POST
/SOAPWebService/services/ImplementacijaSpletneStoritve.
   ImplementacijaSpletneStoritveHttpSoap11Endpoint/ HTTP/1.1
Host: 192.168.1.5:8080
Content-Type: text/xml;charset=UTF-8
Content-Length: 340
SOAPAction: "urn:pridobiDatotekoX"

<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
   soap/envelope/" xmlns:vmes="http://vmesniki.soap.diploma.
   com">
   <soapenv:Header/>
   <soapenv:Body>
      <vmes:pridobiDatotekoX>
         <vmes:velikost>1000</vmes:velikost>
      </vmes:pridobiDatotekoX>
   </soapenv:Body>
</soapenv:Envelope>
```

Seznam 3.1: Primer sporočila SOAP.

Seznam 3.1 prikazuje primer sporočila SOAP, ki je kodirano kot dokument XML. Sestavlja ga ovojnica SOAP `<Envelope>`, ki predstavlja korenski element vsakega sporočila SOAP in vsebuje dva podelementa; neobvezno zaglavje SOAP `<Header>` in obvezno telo SOAP `<Body>`. Zaglavje SOAP se uporablja za prenašanje podatkov povezanih z aplikacijo, ki se bodo obdelali na končnih točkah SOAP vzdolž sporočilne poti, telo SOAP pa vsebuje podatke XML, ki so namenjeni končnemu prejemniku sporočila. Za poročanje napak se uporablja element napaka SOAP `<Fault>`, ki je podelement telesa SOAP [7]. Vsako sporočilo lahko vsebuje samo en element za opis napak. Sporočila SOAP lahko prenašajo tudi priloge, kar je še posebej koristno pri spletnih storitvah, kjer imamo opravka s prilogami.

3.2 WSDL

Odjemalec SOAP je zasnovan na podlagi poznavanja naslova URL, ki določa strežnikovo končno točko, in naslova vrat. WSDL (*angl.* Web Service Description Language) je namenjen opisovanju razpoložljivih spletnih storitev, ki so predstavljene kot množice končnih točk, med katerimi se prenašajo sporočila, ki lahko vsebujejo dokumentno ali postopkovno usmerjene informacije. Tako kot SOAP tudi WSDL temelji na XML [8]. Operacije in sporočila končnih točk so opisane abstraktno in nato povezane na konkreten omrežni protokol in obliko sporočil. Povezane konkretne končne točke so združene v abstraktne končne točke (storitve). WSDL je razširljiv, da omogoča opis končnih točk in njihovih sporočil neodvisno od oblike sporočila ali uporabljenih omrežnih protokolov, ki se uporabljajo za komunikacijo.

```
<wsdl:message name="pridobiDatotekoXRequest">
   <wsdl:part name="parameters" element="ns:pridobiDatotekoX
      "/>
</wsdl:message>

<wsdl:message name="pridobiDatotekoXResponse">
   <wsdl:part name="parameters" element="ns:
      pridobiDatotekoXResponse"/>
</wsdl:message>

<wsdl:portType name="ImplementacijaSpletneStoritvePortType">
   <wsdl:operation name="pridobiDatotekoX">
      <wsdl:input wsaw:Action="urn:pridobiDatotekoX"
                  message="ns:pridobiDatotekoXRequest"/>
      <wsdl:output wsaw:Action="urn:pridobiDatotekoXResponse"
                   message="ns:pridobiDatotekoXResponse"/>
   </wsdl:operation>
</wsdl:portType>
```

Seznam 3.2: Poenostavljen izsek dokumenta WSDL.

Seznam 3.2 prikazuje poenostavljen izsek dokumenta WSDL, sicer pa celoten dokument sestavljajo naslednji elementi [9]:

- `<definitions>` – korenski element dokumenta WSDL, ki običajno opredeljuje imenska področja.
- `<import>` – omogoča uvažanje drugih elementov.
- `<types>` – opredeljuje podatkovne tipe, ki se bodo uporabljali pri izmenjavi sporočil. Za večjo neodvisnost od platforme WSDL za opredelitev podatkovnih tipov uporablja sintakso XML. Podatkovne tipe je dobro opredeliti v ločenih shemah in jih vključiti v dokument WSDL.
- `<message>` – opredeljuje podatkovne elemente operacije. Vsako sporočilo je sestavljeno iz enega ali več delov, ki jih lahko primerjamo s parametri funkcije pri tradicionalnih programskih jezikih. Sporočil je običajno več; za vsako dvosmerno operacijo imamo vhodno in izhodno sporočilo.
- `<portType>` – predstavlja najpomembnejši element, ki opiše vmesnik spletne storitve in množico operacij za določeno končno točko. Vsaka operacija uporablja sporočila. Pri tradicionalnih programskih jezikih ga lahko primerjamo s funkcijsko knjižnico ali razredom.
- `<operation>` – operacija je lahko enosmerna `<input>`, zahteva/odgovor `<input>`, `<output>`, spodbujen odgovor `<output>`, `<input>` ali obvestilo `<output>`.
- `<binding>` – določa konkreten protokol in obliko za podatkovne tipe operacij in sporočil (SOAP, HTTP, MIME).
- `<service>` – določa fizično lokacijo, na kateri se nahaja spletna storitev. Vsebuje lahko eno ali več končnih točk.

3.3 Prednosti in slabosti

3.3.1 Prednosti

SOAP je dobro poznana tehnologija, ki se uspešno uporablja pri razvoju storitveno usmerjene arhitekture (*angl.* Service-oriented architecture) [10]. Številna podjetja so z njegovo pomočjo vzpostavila različne ravni storitev, ki aplikacijam omogočajo dostop do storitev znotraj in zunaj požarnega zidu.

SOAP je bil zasnovan tako, da izkorišča različne načine za prenos podatkov, vključno s sinhronim HTTP/HTTPS, asinhronimi vrstami in celo preko elektronske pošte. Kot zanimivost omenimo še, da je bil SOAP razvit pred prihodom mobilnih naprav in tablic.

3.3.2 Slabosti

- **Nadzor nad spremembami** – spreminjanje spletnih storitev, ki temeljijo na protokolu SOAP, pogosto zahteva spreminjanje odjemalca. V primeru, da je odjemalec spletna aplikacija, spremembe niso nujno problematične, pri mobilni aplikaciji pa se pojavijo težave s posodobitvami.
- **Zapletenost** – izdelava odjemalca SOAP na podlagi dokumenta WSDL je lahko precej zapletena. Dodaten problem se pojavi, ko morajo podjetja in razvijalci isto aplikacijo podpreti na različnih mobilnih platformah. Večkratno spreminjanje zapletenih vmesnikov SOAP je časovno zahtevno in povečuje verjetnost za napake v programski kodi.
- **Ponovna uporaba** – novodobne spletne aplikacije so razvite z uporabo arhitekture REST. Možnosti za njihovo ponovno uporabo bodo zelo omejene, če bodo mobilne aplikacije razvite z uporabo protokola SOAP. S prehodom mobilnih aplikacij na HTML5 ponovna uporaba pridobiva na pomembnosti.

3.3.3 Zmotna prepričanja

- **SOAP je bolj varen** – ta predpostavka se pogosto pojavi zaradi varnostnih metod, ki so del specifikacije *WS-Security*. Zavedati se je namreč potrebno, da je bil razlog za nastanek te specifikacije predvsem neodvisnost protokola SOAP od prenosne poti, kar je onemogočalo predpostavke o razpoložljivi varnosti na prenosni plasti. Pri arhitekturi REST se namreč predpostavlja, da bo prenos opravljen preko HTTP ali HTTPS in bodo na voljo vgrajeni varnostni mehanizmi.

- **Oddaljena naprava je vredna zaupanja** – pred prihodom mobilnih naprav in tablic so do spletnih storitev SOAP dostopali samo aplikacijski strežniki, ki so potrebovali storitve drugih entitet. Običajno sta oba udeležena strežnika veljala za varna. Pri mobilnih aplikacijah je prepričanje, da je naprava vredna zaupanja, zmotno.

Poglavje 4

REST

REST predstavlja arhitekturo, kjer je vsak vir predstavljen kot spletna storitev z enoznačnim naslovom URL. REST sam po sebi ni standardiziran protokol, opredeljuje pa, kako uporabljati obstoječe standarde. Načelo REST je uporaba protokola HTTP, tako kot je modeliran [11]. Za dostopanje in spreminjanje virov se tako uporabljajo standardne metode HTTP: GET, PUT, POST in DELETE. Na enak način delujejo tudi spletni brskalniki, ki z uporabo naslovov URL izvajajo zahteve na strežniku, ki kot odgovor vrne datoteko HTML [12]. Seveda brskalniki ne zahtevajo zgolj datotek HTML, ampak tudi slogovne datoteke CSS, slike in številne druge tipe datotek. Zaradi svoje prilagodljivosti je protokol HTTP primeren tudi za izmenjavo podatkov, ki nimajo vizualne komponente. Arhitektura REST tako izkorišča razširjenost protokola HTTP in njegove zmožnosti, zaradi česar je odličen kandidat za prenos podatkov.

Pri izdelavi nabora storitev REST se spodbuja uporaba različnih metod HTTP, ki so poimenovane kot glagoli, ki opredeljujejo storitev. Pregled elementov določene tabele tako dosežemo z zahtevo GET, medtem ko za dodajanje novih elementov uporabimo zahtevo POST. Pri arhitekturi REST so vse zahteve HTTP neodvisne ena od druge. Vsaka zahteva odjemalca mora biti opremljena z vsemi podatki, ki jih potrebuje strežnik.

Tabela 4.1 prikazuje šest storitev. Vsaka ima opisno ime, naslov URL, na

Ime storitve	**Naslov URL**	**Metoda**	**Parameter**
Seznam vseh datotek	http://streznik/datoteka	GET	/
Prenos ene datoteke	http://streznik/datoteka/1	GET	ID datoteke
Posodabljanje ene datoteke	http://streznik/datoteka/1	PUT	ID datoteke
Dodajanje ene datoteke	http://streznik/datoteka	POST	/
Brisanje vseh datotek	http://streznik/datoteka	DELETE	/
Brisanje ene datoteke	http://streznik/datoteka/1	DELETE	ID datoteke

Tabela 4.1: Primer spletne storitve REST.

Operacije CRUD	**Metode REST (HTTP)**	**Operatorji SQL (podatkovna baza)**
CREATE – ustvari ali doda nov vnos	POST	INSERT
READ – branje in pridobivanje podatkov	GET	SELECT
UPDATE – posodobi ali uredi obstoječe podatke	PUT	UPDATE
DELETE – brisanje obstoječih podatkov	DELETE	DELETE/DROP

Tabela 4.2: Primerjava operacij CRUD, metod REST in operatorjev SQL.

katerem se nahajajo viri, metodo HTTP za izvedbo dejanja in opombo, kjer so navedeni morebitni parametri. Ta zelo preprost primer prikazuje, kako lahko uporabimo metode HTTP za opis številnih funkcij. Z uporabo štirih različnih metod in enega naslova URL smo opredelili celoten nabor storitev CRUD (*angl.* Create Read Update Delete) za eno vrsto podatkov. Če bi potrebovali podoben nabor storitev za drugo vrsto podatkov, bi preprosto opredelili nov naslov URL.

Tabela 4.2 prikazuje primerjavo operacij CRUD, metod REST in operatorjev SQL. V nekaterih primerih REST preprosto ni dovolj, da bi z njim lahko opisali vsako storitev. Dober primer so transakcije, ki jih ne moremo

opisati zgolj s štirimi metodami.

4.1 JSON

V prejšnjem poglavju smo opisali, kako REST opredeljuje storitve, ne pa tudi, kaj je bilo dejansko poslano in prejeto, ko so bile uporabljene. Pravzaprav ni enotnega odgovora na to vprašanje, ker REST ne narekuje, kako naj odjemalec ali strežnik izvedeta določeno nalogo. Nekatere storitve bodo tako vračale besedilo, druge pa binarne podatke, kot na primer spletni brskalnik, ko zahteva sliko.

JSON (*angl.* JavaScript Object Notation) je preprosta oblika za izmenjavo podatkov, ki je neodvisna od programskega jezika. Zaradi besedne zasnove je enostaven za branje in pisanje tako ljudem kot tudi računalnikom [13]. Uporablja se predvsem za izmenjavo podatkov med strežnikom in spletno aplikacijo. V primerjavi z XML je JSON manjši in hitrejši. Njegova sintaksa je enaka tisti za ustvarjanje objektov *JavaScript*. Zaradi tega lahko program *JavaScript* uporabi vgrajeno funkcijo `eval()`, ki iz podatkov JSON naredi objekte *JavaScript*. Na ta način se lahko izognemo uporabi posebnih razčlenjevalnikov.

```
{
"id":1,
"podatki":"**********"
}
```

Seznam 4.1: Primer objekta JSON.

Seznam 4.1 prikazuje primer zelo preprostega objekta JSON z dvema paroma ključ/vrednost. Prvi ima ključ `id` z vrednostjo `1`, drugi pa ima ključ `podatki` z vrednostjo `**********`. Objekti JSON so tako zbirke parov ključ/vrednost [12]. Na Seznamu 4.1 sta vrednosti obeh ključev preprosta tipa. Za vrednost ključa lahko uporabimo tudi drug objekt ali celo matriko.

```
{
"ime":"Matjaz",
```

```
"priimek":"Rajnar",
"starost":24,
"sport":["fitnes", "kolesarjenje", "plavanje"],
"naslov":{
        "ulica":"Borovnjakova ulica"
        "hisnaStevilka":28
        "posta":9000
        "mesto":"Murska Sobota"
},
"telefonskaStevilka":[
        {
                "vrsta":"mobilni",
                "stevilka":"031 XXX XXX"
        }
        {
                "vrsta":"domaci",
                "stevilka":"02 XXX XX XX"
        }
]
}
```

Seznam 4.2: Primer kompleksnejšega objekta JSON.

Seznam 4.2 prikazuje primer kompleksnejšega objekta JSON. Vrednosti ključev `ime`, `priimek` in `starost` so preprostega tipa, vrednost ključa `naslov` pa je drug objekt JSON, ki vsebuje lastne pare ključ/vrednost. Ključ `sport` ustreza matriki nizov, medtem ko ključ `telefonskaStevilka` ustreza matriki objektov.

4.2 Prednosti in slabosti

4.2.1 Prednosti

- **Preprostost** – REST je bil zasnovan za delo na odjemalcih, ki imajo omejene vire, od spletnih brskalnikov, različnih vrst omrežne opreme, do množice različnih vrst storitev [10]. Zaradi zahteve po preprosti in

prilagodljivi implementaciji je zelo preprost in podoben veliki večini omrežnega prometa.

- **Prijaznost do infrastrukture** – požarni zidovi in strežniki so že optimizirani za arhitekturo REST, ker so optimizirani za promet HTTP/HTTPS.
- **Možnost predpomnjenja** – nekatere vrste zahtev REST so primerne za predpomnjenje. Za primer vzemimo zahteve po slikah, kjer je celotna zahteva predstavljena kot naslov URL. Omrežna infrastruktura namenjena predpomnjenju se lahko odzove s pravilnim odgovorom.
- **Razširljivost** – vsaka zahteva REST je neodvisna od ostalih in se obdela na podlagi podatkov, ki jih vsebuje. To omogoča veliko razširljivost.
- **Učinkovitost** – medtem ko spletne storitve SOAP vedno vračajo dokumente XML, spletne storitve REST omogočajo prilagodljivost glede vrste podatkov, ki jih vračajo. Privzeta oblika za prenašanje koristne vsebine so objekti JSON. Prve različice mobilne platforme Android niso vključevale ogrodja za razčlenjevanje objektov JSON.

4.2.2 Slabosti

- **Varnost je potrebno vključiti že pri načrtovanju** – kot pri vsakem podatkovnem protokolu, mora biti tudi pri arhitekturi REST varnost dobro načrtovana. V mislih je potrebno imeti podatke mobilne aplikacije, saj se z naraščanjem števila mobilnih aplikacij povečuje tudi možnost napadov in zlorab. Skrbno je potrebno preučiti, kateri podatki se bodo prenašali med odjemalcem in spletno storitvijo. Informacije o identiteti uporabnika se naj ne bi prenašale na mobilno napravo, razen če je to nujno potrebno.
- **Ni rigiden standard** – obstajajo številne implementacije, označene kot spletne storitve REST, ki v resnici niso nič drugega kot navaden *servlet*, ki nadzira objavljeno koristno vsebino in izvaja njene operacije. To je pomanjkljiv pristop, ki ne izkorišča številnih prednosti arhitekture REST. Vključitev ukaznih parametrov v zahtevo naslova URL omogoča

spremljanje sistema in upravljanje zahteve. Uporaba celotnega nabora metod HTTP (POST, GET, PUT, DELETE) omogoča funkcionalno ločitev spletne storitve na različne dele.

Poglavje 5

SOAP, REST in Android

Velika večina današnjih spletnih strani uporablja spletne storitve REST. Omenjen pristop na primer uporabljajo Facebook, Twitter in Amazon [14]. Zadnjih nekaj let so še posebej priljubljene oblačne arhitekture, ki svoje spletne storitve v večini primerov ponujajo preko vmesnikov, ki temeljijo na arhitekturi REST. Ne smemo pa pozabiti na dejstvo, da še vedno obstaja ogromno spletnih storitev, ki temeljijo na protokolu SOAP.

5.1 Uporaba SOAP na Androidu

Google do danes na svoji mobilni platformi Android ni zagotovil podpore za neposredno klicanje spletnih storitev, ki temeljijo na protokolu SOAP [15]. Mobilne naprave se danes soočajo predvsem z omejeno pasovno širino, viri in kapaciteto baterije, kar zaradi same besedne zasnove XML in ponavljanja, vnaša številne težave pri njegovi uporabi. Neučinkovitost XML se tako kaže v besedni serializaciji števil, zaključnih značk in imenskih prostorov. SOAP se sooča z enakimi težavami, dodaja pa nekaj novih. Razvijalci lahko vključijo svoje lastne knjižnice ali uporabijo obstoječe knjižnice tretjih oseb, ki omogočajo razvoj odjemalcev SOAP na Androidu.

5.1.1 Ročna gradnja sporočil SOAP

Kljub temu da Android ne zagotavlja podpore za neposredno klicanje spletnih storitev SOAP, vgrajena podpora za XML in HTTP omogoča ročno gradnjo sporočil SOAP in odpremo zahtev preko razreda `HttpClient`. Odgovor se nato ročno razčleni in pretvori v javanske objekte. Zaradi omejenih virov pri mobilnih napravah sta ročna gradnja in razčlenjevanje sporočil SOAP zelo dobrodošla pri zmanjšanju količine pomnilnika in zahtev, ki jih je potrebno obdelati. Prav tako so mobilne aplikacije veliko bolj usmerjene v opravljanje določene naloge v primerjavi z namiznimi in spletnimi aplikacijami, kar še dodatno prispeva k manjši kompleksnosti pri ročni gradnji sporočil SOAP. Kljub naštetim prednostim je izdelava lastne knjižnice SOAP časovno zahtevna in ni najbolj praktična, zato se večina razvijalcev poslužuje obstoječih knjižnic.

5.1.2 Obstoječe knjižnice za podporo SOAP

V nadaljevanju si bomo ogledali dve prilagoditvi protokola SOAP, in sicer knjižnico SOAP za Android, ki se imenuje **kSOAP2 Android**, in nabor optimizacijskih tehnik, ki jih s skupnim imenom označujemo kot **WSOAP** (*angl.* Wireless SOAP).

Knjižnica kSOAP2 Android

Knjižnica kSOAP2 Android prvotno ni bila razvita za mobilne naprave Android. Vse skupaj se je začelo s knjižnico **kSOAP** davnega leta 2001 [14]. V primerjavi z osebnimi računalniki so bile mobilne naprave desetletje nazaj zelo omejene z viri in razvoj aplikacij z uporabo obstoječih tehnologij je bil praktično nemogoč. Največjo težavo je predstavljal zelo majhen pomnilnik in sama velikost paketov pri uporabi XML in SOAP. Mobilne naprave preprosto niso bile dovolj zmogljive, da bi lahko delale s knjižnicami, ki so bile prvotno razvite za osebne računalnike.

kSOAP: Kot smo že omenili, so XML in SOAP paketi običajno precej

veliki in vsebujejo na stotine razredov. Poleg tega so odvisni od izvajalnega okolja *Java*, ki na mobilnih napravah ni na voljo. Zaradi velikega števila mobilnih naprav je bila želja razvijalcev, da bi lahko arhitekturo spletnih storitev nemoteno vključili tudi na njih. V okviru odprtokodnega projekta *Enhydra*, pod vodstvom Stefana Hausteina, sta bili razviti knjižnici kXML in kSOAP, z namenom omogočiti aplikacije XML in SOAP na manj zmogljivih mobilnih napravah. Obe knjižnici sta zelo dobro dokumentirani in združeni v eni sami `.jar` datoteki, ki zaseda manj kot 42 kB prostora [16].

Knjižnica kSOAP predstavlja implementacijo SOAP, ki temelji na *XML-RPC*. Za njeno uporabo sta potrebna razred `HttpConnection` in njegova metoda `call()` [17]. Za razliko od običajnih vmesnikov API, kXML in kSOAP skupaj ponujata učinkovito okolje za razvoj spletnih storitev na mobilnih napravah.

Kljub vsem funkcijam za implementacijo spletnih storitev na mobilnih napravah se okolje na njih še vedno precej razlikuje od tistega, ki ga poznamo pri osebnih računalnikih, tako da je uporaba knjižnice namenjena manj zahtevnim mobilnim aplikacijam.

kSOAP2: Knjižnica kSOAP2 predstavlja popolno prenovo knjižnice kSOAP. Oglejmo si nekaj najpomembnejših izboljšav:

- izboljšana zgradba knjižnice,
- izboljšana podpora kodiranju v stilu *literal*,
- podpora serializaciji SOAP je sedaj opcijska in se nahaja v ločenem paketu,
- ločeni razredi so sedaj združeni v en sam razred `SoapSerializationEnvelope`, ki zagotavlja podporo serializaciji SOAP; razred `SoapSerializationEnvelope` razširja osnovni razred `SoapEnvelope`,
- zastavico *.NET* lahko uporabimo za preklop `SoapSerializationEnvelope` iz običajnega načina v upravljanje z imenskimi prostori, kar je privzeti način v okolju *.NET*.

Leta 2007 je Android razvijalec z imenom Jorge Jimenez v skupini Android Developers objavil množico popravkov za kSOAP2, ki so se nanašali na

mobilno platformo Android [14]. Nedolgo za tem se je pojavila nova knjižnica z imenom kSOAP2 Android, ki je dostopna na naslovu http://code.google.com/p/ksoap2-android/. Gre za dovršeno, prilagodljivo, odprtokodno in popolnoma brezplačno knjižnico. Skupnost razvijalcev je za dodatne informacije in podporo naredila nove sezname za elektronsko pošto in spletno stran *wiki*. Tako je knjižnica kSOAP2 Android postala knjižnica SOAP za mobilne naprave Android. V času pisanja diplomskega dela je zadnja stabilna različica knjižnice 3.1.1.

Knjižnica WSOAP

Knjižnica WSOAP je nabor optimizacijskih tehnik, ki temelji na dveh pristopih:

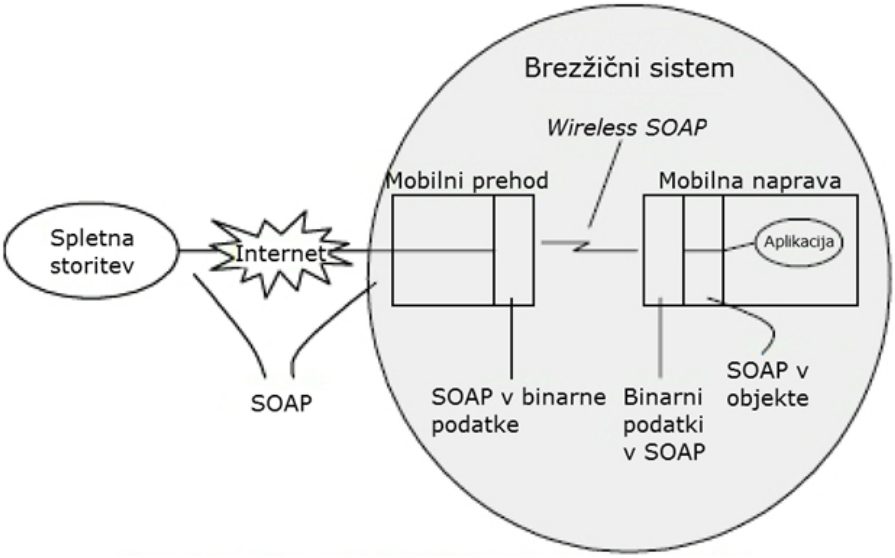
- enakovrednost imenskih prostorov (*angl.* Name Space Equivalency) in
- poznavanje dokumenta WSDL (*angl.* WSDL Aware Encoding).

Medtem ko so predhodne tehnike poizkušale kodirati ali stiskati sporočila SOAP, da bi omogočile obnovo v nespremenjeni obliki, zdaj to ni več potrebno in zadošča obnova v enakovredni obliki [18]. Enakovrednost imenskih prostorov izkorišča dejstvo, da je v kateremkoli danem dokumentu izbira določene predpone za označevanje povezave z imenskim prostorom poljubna. V dokumentu sta oznaki `<soap:Envelope>` in `<s0:Envelope>` enakovredni, dokler sta predponi `soap` in `s0` povezani z enakim imenskim prostorom.

V primeru, da mobilni prehod in mobilna naprava poznata dokument WSDL, lahko znatno prihranimo na pasovni širini. Za vsak vmesnik spletne storitve, ki ga uporablja odjemalec, lahko analiziramo dokument WSDL in ustvarimo potrebne kodirne tabele. Z uporabo preprostega sinhronizacijskega protokola med mobilnim prehodom in mobilno napravo, se sinhronizacija običajno izvede samo enkrat.

Osnovna načela WSOAP so:

- zagotoviti statično kodiranje, ki temelji na shemi SOAP,

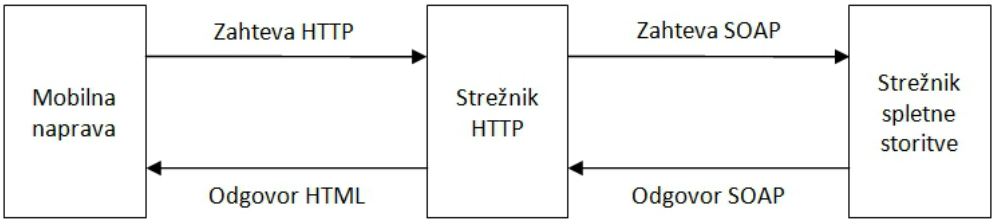


Slika 5.1: Primer vpeljave WSOAP [18].

- izkoristiti opis storitev WSDL za izdelavo prilagojenega kodiranja za vmesnike spletnih storitev,
- zahteva po funkcionalni enakosti namesto obnove v nespremenjeni obliki,
- omejiti računske stroške z dajanjem prednosti kodiranju in iskanjem s pomočjo računanja, kjer je to mogoče,
- kodiranje z uporabo binarnega sistema.

Za boljše razumevanje si oglejmo primer vpeljave WSOAP na Sliki 5.1, kjer se mobilna naprava poveže preko brezžične povezave z mobilnim prehodom, ki se nato poveže naprej preko žičnega omrežja na spletno storitev. Delovanje WSOAP je omejeno samo na brezžično povezavo.

Z uporabo navedenih tehnik lahko zmanjšamo velikost sporočila od 3- do 12-krat v primerjavi z navadnim protokolom SOAP in tako znatno prihranimo na pasovni širini, kar je pri mobilnih napravah še posebej dobrodošlo. Posledično imamo tudi manj izgubljenih sporočil in prihranimo življenjsko dobo baterije za druge operacije.



Slika 5.2: Prikaz HTML frontend.

5.1.3 Druge možnosti za dostop do spletnih storitev

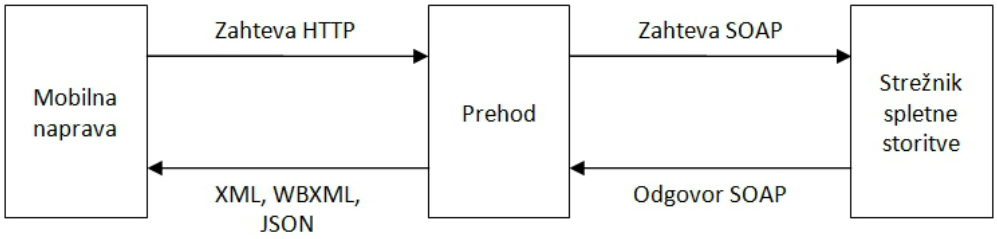
Namesto uporabe obstoječih knjižnic za podporo SOAP lahko uporabimo tudi druge pristope odjemalec-strežnik, ki so še posebej primerni za mobilne naprave [19].

HTML frontend

HTML frontend predstavlja vmesnik med uporabnikom in zalednim sistemom — Slika 5.2. Na strežniku HTTP se izvaja spletna aplikacija, ki zahteve z mobilne naprave posreduje v nadaljnjo obdelavo strežniku spletnih storitev, kot odgovore pa vrača dokumente HTML. Mobilna naprava za dostop do spletne aplikacije in komunikacijo uporablja spletni brskalnik. Kot je prikazano na sliki, so zahteve med mobilno napravo in strežnikom HTTP navadne zahteve HTTP.

HTML frontend lahko izdelamo hitro in enostavno. Spletne aplikacije se lahko izvajajo na kateremkoli spletnem strežniku, na voljo pa jim je tudi nabor ogrodij za razvoj odjemalcev SOAP. Ta pristop je neodvisen od naprave in vsaka mobilna naprava, ki ima spletni brskalnik, lahko dostopa do iste spletne aplikacije.

Z uporabo vmesnika spletnega brskalnika na mobilni napravi ni mogoče uporabiti določenih funkcij naprave. Tudi sinhronizacija z aplikacijami, kot so koledar, seznam stikov, zemljevidi in sporočili SMS, ni mogoča. Za delovanje spletne aplikacije mora mobilna naprava imeti vzpostavljeno povezavo



Slika 5.3: Prikaz prehoda za spletno storitev.

z internetom. V primeru, da se povezava prekine, spletna aplikacija ni več uporabna in obstaja velika nevarnost izgube podatkov. Spletne aplikacije, ki uporabljajo HTML frontend, morajo ob spremembi določenega dela spletne strani znova naložiti celotno spletno stran.

Prehod za spletno storitev

Med mobilno napravo in spletno storitvijo se nahaja prehod, ki zahteve z mobilne naprave pretvori v klice spletne storitve — Slika 5.3. Odgovori spletne storitve so nato pretvorjeni v preprostejšo obliko in posredovani prehodu spletne storitve. Ta jih nato pretvori v obliko XML, WBXML ali JSON in posreduje mobilnemu odjemalcu. Ena zahteva posredovana prehodu spletne storitve lahko sproži več zahtev in odgovorov med prehodom in spletno storitvijo.

Z uporabo prehoda lahko razvijamo bolj optimizirane vmesnike za mobilne naprave, ne da bi pri tem bilo potrebno spreminjati spletne storitve, ki se izvajajo na svojem strežniku. Zaradi preprostejše oblike podatkov in optimizacije se zmanjša količina prenesenih podatkov in zahtev, ki jih je potrebno obdelati. To izboljša odzivnost aplikacij in zmanjša zakasnitve.

Uporaba prehoda predstavlja dodaten element, ki ga je potrebno vzdrževati. Vsaka nova funkcionalnost, ki jo želimo dodati, zahteva spremembe pri strežniku spletne storitve, prehodu in mobilni napravi.

5.2 Uporaba REST na Androidu

Za razliko od spletnih storitev SOAP, za dostopanje do spletnih storitev REST ne potrebujemo dodatnih knjižnic, ampak uporabimo kar knjižnico **Android HttpClient**, ki je razvijalcem na voljo že od zgodnjih različic mobilne platforme Android. Spletne storitve REST so še posebej primerne za zagotavljanje vsebine mobilnim napravam in tablicam, ker zahtevajo zelo malo virov. Klic spletne storitve REST iz mobilne aplikacije zahteva izvedbo zahteve HTTP v ozadju in posredovanje rezultatov niti uporabniškega vmesnika.

Arhitekturo REST lahko na mobilnih napravah Android vključimo na naslednje načine [20]:

- z uporabo vmesnika `Service`,
- z uporabo vmesnika `Content Provider`,
- z uporabo vmesnika `Content Provider`, ki ga razširimo z vmesnikom `Sync Adapter`.

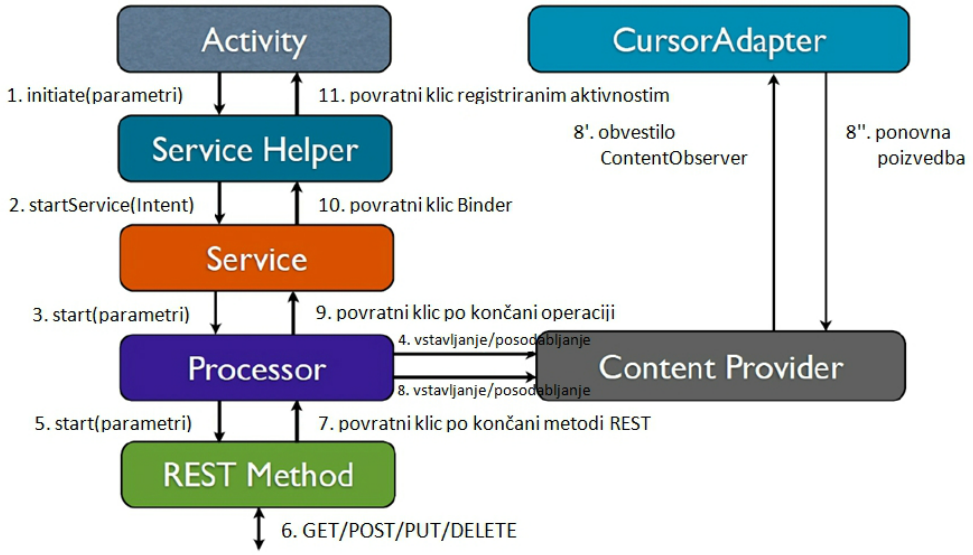
Zavedati se je potrebno, da obstajajo tudi drugi načini, ki jih razvijalci lahko uporabijo. Tisti z dovolj izkušnjami lahko razvijejo tudi svoje lastne načine, v kolikor so v skladu z osnovnimi načeli delovanja mobilne platforme Android.

5.2.1 Vmesnik Service

Izkorišča storitve, ki so sestavni del mobilne platforme Android in niso v interakciji z uporabnikom. Za boljše razumevanje dogajanja na Sliki 5.4 bomo uporabili pristop od spodaj navzgor in si ogledali vlogo posameznih elementov.

Metoda REST (*angl.* REST Method):

- pripravi naslov URL in telo v zahtevi HTTP,
- izvede transakcijo HTTP,



Slika 5.4: Prikaz delovanja vmesnika Service [20].

- obdela odgovor HTTP, ki ga prejme od strežnika,
- izbere optimalno obliko za odgovor (JSON, XML ali binarno),
- omogoči stiskanje vsebine z uporabo knjižnice **gzip**, kjer je to mogoče,
- izvede metodo REST v okviru delovne niti (*angl.* Worker Thread),
- uporabi odjemalca `Apache HTTP`.

Številni vmesniki API za podporo arhitekturi REST uporabniku omogočajo ročno izbiro oblike za odgovor. Priporoča se uporaba oblike JSON, ker se izvaja veliko hitreje kot XML. Knjižnica gzip je namenjena stiskanju vsebine. Njena uporaba pospeši prenos podatkov in prihrani življenjsko dobo baterije za druge operacije. Metoda REST se mora izvesti v okviru delovne niti, ker izvedba transakcije HTTP traja nekaj časa in lahko pride do poteka časovne omejitve.

Procesor (*angl.* Processor) je namenjen preslikovanju stanja strežniških virov v lokalno podatkovno bazo. Vsaka vrstica podatkovne baze predstavlja natanko en vir. Za izboljšano uporabniško izkušnjo je dobro prikazati stanje

aplikacije, kar dosežemo z zastavicami. Zanima nas, kdaj je aplikacija sredi izvajanja metode POST za določen vir, kdaj posodablja ali briše določen vir in kdaj je sredi izvajanja transakcije HTTP.

Ločimo naslednje primere:

- V primeru metode POST procesor s klicem operatorja INSERT vstavi novo vrstico v podatkovno bazo, tako da ta ustreza viru, ki ga želimo dodati. Preden izvedemo metodo REST, ustrezno nastavimo zastavice za določen vir. Ko se metoda REST uspešno izvede, je potrebno posodobiti vrstico v podatkovni bazi s pomočjo vmesnika `Content Provider` in odstraniti zastavice, ki smo jih nastavili v prejšnjem koraku.
- Na podoben način deluje tudi metoda PUT, kjer procesor s klicem operatorja UPDATE posodobi obstoječo vrstico v podatkovni bazi. Preden izvedemo metodo REST, ustrezno nastavimo zastavice za določen vir. Ko se metoda REST uspešno izvede, je potrebno posodobiti vrstico v podatkovni bazi in odstraniti zastavice, ki smo jih nastavili v prejšnjem koraku.
- Metoda DELETE je zelo preprosta. Preden izvedemo metodo REST, ustrezno nastavimo zastavice za določen vir. Tega še ne bomo izbrisali, ker se metoda REST sploh še ni izvedla. Ko se metoda REST uspešno izvede, lahko izbrišemo vrstico v podatkovni bazi.
- Metoda GET je najpreprostejša. Pred izvedbo metode REST procesor ne počne nič. Ko se metoda REST uspešno izvede, procesor vstavi nove vire v podatkovno bazo.

Storitev (*angl.* The Service) je zasnovana za izvajanje operacij v ozadju, in sicer v okviru glavne niti, tudi ko se je aktivnost že zaključila. Za izvedbo operacij moramo znotraj storitve ustvariti tudi delovno nit. Vmesnik storitve implementiramo z uporabo sporočilnih objektov (*angl.* Intents). Ko storitev prejme sporočilni objekt, najprej razširi njegovo vsebino, nato pa izvede klic

funkcije *Java*, ki se bo izvedla na procesorju. Na poti nazaj storitev od procesorja prejme povratni klic o dokončani operaciji, njeno stanje pa se z novim povratnim klicem posreduje višje ležečemu elementu.

Na mobilnih napravah se hkrati izvaja ogromno procesov. Zamislimo si aplikacijo za dostop do priljubljenega družabnega omrežja. Zaradi hitrega pomikanja navzdol po seznamu novic mora aplikacija hkrati izvesti veliko število prenosov slik, kar slabo vpliva na ostale aplikacije, ki prav tako potrebujejo dostop do spleta. Da bi rešili težavo, za omenjeno storitev implementiramo vrsto, ki dovoljuje samo omejeno število hkratnih prenosov slik.

Ko se vse operacije izvedejo, je storitev potrebno zaustaviti. V primeru, da tega ne storimo, bo operacijski sistem predpostavljal, da se aplikacija še vedno izvaja in ne bo sprostil pomnilnika in ostalih virov, kar ponovno slabo vpliva na ostale aplikacije.

Pomočnik storitve (*angl.* The Service Helper):

- nudi preprost asinhroni vmesnik, ki se uporablja skupaj z uporabniškim vmesnikom,
- preveri, če se metoda REST še vedno izvaja,
- ustvari sporočilni objekt,
- določi vrsto operacije in enoznačen ID zahteve,
- določi parametre značilne za določeno metodo REST,
- izvede klic funkcije, ki sporočilni objekt posreduje storitvi,
- vrne enoznačen ID zahteve,
- prejme povratni klic od storitve in ga posreduje poslušalcem (*angl.* Listeners) uporabniškega vmesnika.

Poslušalci so običajno aktivnosti, ki potrebujejo rezultate določenih metod REST. Podatkov, ki se prenašajo med storitvijo in pomočnikom storitve pri povratnem klicu, mora biti čim manj.

Aktivnost (*angl.* The Activity) je pravzaprav uporabniški vmesnik naše aplikacije, ki ga uporabnik lahko kadarkoli zapre, operacija pa nadaljuje svoje

izvajanje v ozadju. V primeru dohodnega klica lahko tudi aktivnost začasno ustavi trenutno aplikacijo.

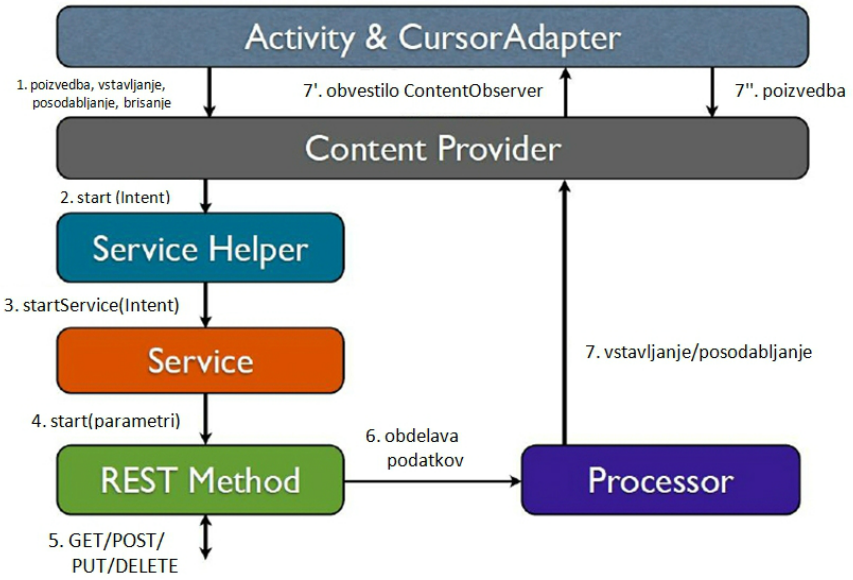
Ločimo naslednje primere:

- Aktivnost sproži klic metode REST in počaka na povratni klic od pomočnika storitve. Uporabniku sporoči ali se je operacija uspešno izvedla ali pa je pri izvedbi prišlo do napak.
- Aktivnost sproži klic metode REST, se začasno ustavi, ponovno zažene, nato pa se pojavi povratni klic. V tem primeru je potrebno shraniti enoznačen ID zahteve, da lahko s pomočnikom storitve ugotovimo, če se metoda REST še vedno izvaja, in počakati na povratni klic.
- Aktivnost sproži klic metode REST in se začasno ustavi. Medtem se metoda REST zaključi, aktivnost pa se ponovno zažene. Tudi v tem primeru je potrebno shraniti enoznačen ID zahteve, da lahko s pomočnikom storitve ugotovimo, če se metoda REST še vedno izvaja. Ker se je njeno izvajanje že zaključilo, lahko njeno stanje ugotovimo iz podatkovne baze.

5.2.2 Vmesnik Content Provider

Vmesnik `Content Provider` omogoča enostavno shranjevanje in pridobivanje podatkov iz podatkovne baze na mobilni platformi Android. Nudi metode za dodajanje, posodabljanje, pridobivanje in brisanje, ki nato dostopajo do podatkovne baze. Zaradi preslikave med metodami REST in nekaterimi metodami vmesnika `Content Provider` je ta primeren za proženje klicev metod REST. Metoda POST REST je tako namenjena dodajanju, metoda PUT REST posodabljanju, metoda GET REST pridobivanju in metoda DELETE REST brisanju podatkov iz podatkovne baze.

Metoda REST, storitev in pomočnik storitve na Sliki 5.5 delujejo enako kot pri prejšnjem načinu, nekoliko pa se razlikuje delovanje procesorja in vmesnika `Content Provider`.



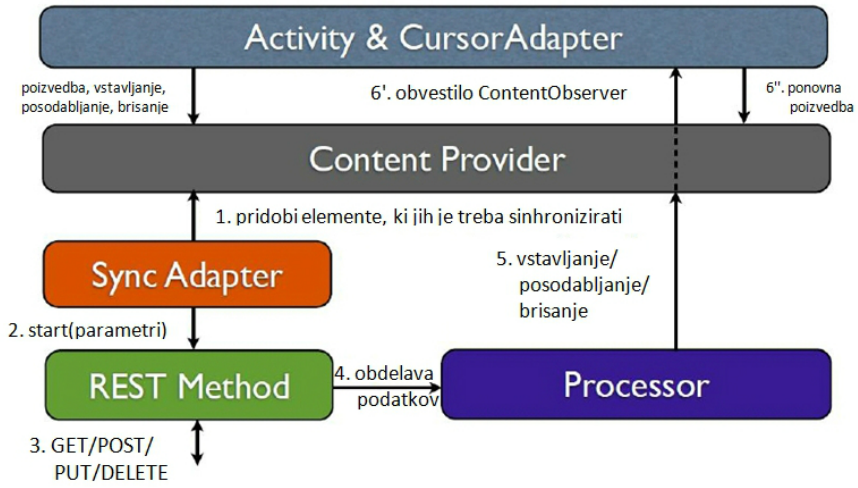
Slika 5.5: Prikaz delovanja vmesnika Content Provider [20].

V primeru, da želimo v podatkovno bazo dodati nov vnos, aktivnost pri vmesniku `Content Provider` najprej poišče metodo, v katero se preslika metoda REST. Ker dodajamo nov vnos, to ustreza metodi POST REST. Aktivnost pripravi tudi vse potrebne podatke in sproži klic metode POST REST.

Vmesnik `Content Provider` nato v podatkovni bazi ustvari novo vrstico, aktivnosti vrne identifikator URI na novo ustvarjene vrstice in ustrezno nastavi zastavice za določen vir, tako da aktivnost na uporabniškem vmesniku lahko prikazuje stanje aplikacije.

Pomočnik storitve nato sproži klic storitve in izvede metodo REST. Po uspešni izvedbi metode REST, procesor odstrani zastavice, ki jih je postavil vmesnik `Content Provider`, posodobi vrstico v podatkovni bazi in aktivnosti sporoči, da je zaključil z delom.

V primeru, da se metoda REST ne izvede uspešno, vmesnik `Content Provider` nastavi opomnik in jo po določenem času znova izvede. Za prihranek baterije se uporabljajo posebni algoritmi, ki eksponentno povečujejo



Slika 5.6: Prikaz delovanja vmesnika Content Provider, ki je razširjen z vmesnikom Sync Adapter [20].

čase med ponovitvami.

5.2.3 Vmesnik Content Provider razširjen z vmesnikom Sync Adapter

Vmesnik `Sync Adapter` omogoča sinhronizacijo oddaljene in lokalne vsebine — Slika 5.6. Mobilna platforma Android uporablja upravitelja sinhronizacij (*angl.* Sync Manager), ki upravlja z vmesniki `Sync Adapter` vseh aplikacij. Upravitelj sinhronizacij implementira vrsto, v katero se shranijo zahteve vmesnikov `Sync Adapter`. Tudi če je vrsta prazna, upravitelj sinhronizacij nikoli ne izvede sinhronizacije neposredno, ker skrbi za ohranjanje celovitosti celotnega sistema.

V primeru, da želimo v podatkovno bazo dodati nov vnos, aktivnost pri vmesniku `Content Provider` najprej poišče metodo, v katero se preslika metoda REST. Ker dodajamo nov vnos, to ustreza metodi POST REST. Aktivnost pripravi tudi vse potrebne podatke in sproži klic metode POST REST.

Vmesnik `Content Provider` nato v podatkovni bazi ustvari novo vrstico, ustrezno nastavi zastavice in pred vrnitvijo sproži zahtevo po sinhronizaciji.

Upravitelj sinhronizacij izvede vmesnik `Sync Adapter`, ki v podatkovni bazi preveri, katere vire je potrebno sinhronizirati. Sledi branje vrstice iz podatkovne baze in izvedba metode REST. Po uspešni izvedbi metode REST, vmesnik `Sync Adapter` posodobi vrstico v podatkovni bazi in aktivnosti sporoči, da je zaključil z delom.

V primeru, da se metoda REST ne izvede uspešno, vmesnik `Sync Adapter` posreduje napake upravitelju sinhronizacij, ki ga, z uporabo posebnih algoritmov za eksponentno povečevanje časa med ponovitvami, postavi na konec vrste.

Poglavje 6

Mobilne aplikacije za dostop do spletnih storitev

Preden se lotimo razvoja aplikacije za mobilne naprave, se postavlja vprašanje, zakaj bi razvijali tako aplikacijo, če storitev, na katero se želimo povezati, že obstaja v prilagojeni obliki? Z drugimi besedami, zakaj preprosto ne uporabimo spletnega brskalnika za povezavo s spletno storitvijo?

Oglejmo si razloge, zakaj je boljše uporabljati mobilne aplikacije skupaj s pristopi, ki smo jih spoznali v predhodnih poglavjih [20]:

- Zmožnost povezave aplikacije z mobilno platformo Android aplikaciji omogoča dostop do vseh zasebnih vmesnikov API. Aplikacija lahko uporablja tudi sporočilne objekte za koordinacijo med funkcijami različnih aplikacij, ki tako dobijo nove funkcionalnosti, in tako obogati celotno mobilno platformo Android.
- Aplikacija se lahko izvaja v ozadju. V primeru, da se na strežniku pojavi nova vsebina, aplikacija to ugotovi in uporabniku prikaže ustrezno obvestilo.
- Aplikacija se izvaja na napravah, ki nimajo zanesljive povezave s spletom. Zaradi tega se nekatere metode ne izvedejo. Aplikacije, ki uporabljajo take metode se običajno lahko izvajajo v ozadju in tako ponovijo spodletele operacije z uporabo opomnikov. Uporabniku tako ni

potrebno ponovno naložiti spletne strani oziroma ponoviti spodletele operacije, kar omogoča hitrejše delovanje aplikacije.

- Aplikacija ima možnost pridobivanja spletne vsebine v številnih oblikah. Vsebina se nato ustrezno obdela in shrani v podatkovno bazo. Ko uporabnik želi pridobiti novo vsebino, ima možnost, da naloži samo vsebino, ki je novejša ali starejša od trenutne. Prav tako se nikoli ne naložijo vsi dokumenti HTML ali kakršnakoli oblika *JavaScript*, ki bi lahko bila prevelika za prenos.
- Uporabniški vmesnik aplikacije je lahko skladen z operacijskim sistemom.

Poglavje 7

Eksperimentalno in razvojno okolje

Slika 7.1 prikazuje eksperimentalno okolje, ki je bilo uporabljeno pri tem diplomskem delu. Mobilne aplikacije so bile najprej preizkušene na navidezni napravi Android, meritve pa so bile opravljene na mobilni napravi Android.

Obe napravi Android se preko brezžične dostopne točke *Innbox V50U* povezujeta na strežnik spletne storitve, ki se nahaja na lokalnem naslovu http://192.168.1.5. Brezžična dostopna točka je obenem tudi usmerjevalnik in optični modem za povezavo z internetom.

7.1 Lastnosti uporabljenih naprav

7.1.1 Mobilna naprava Android

Razvite aplikacije so se izvajale na mobilni napravi *LG Optimus 4X HD P880*. Lastnosti naprave, ki so pomembne za našo raziskavo, so povzete v Tabeli 7.1.

Slika 7.1: Uporabljeno eksperimentalno okolje.

Strojna oprema	Procesor	Quad-core 1.5 GHz Cortex-A9
	Delovni pomnilnik	1024 MB
	Notranji pomnilnik	16 GB
	Brezžično omrežje	Wireless 802.11 a/b/g/n
Programska oprema	Android verzija	4.1.2
	Različica jedra	3.1.10

Tabela 7.1: Lastnosti mobilne naprave Android.

Strojna oprema	Procesor	ARM (armeabi-v7a)
	Delovni pomnilnik	512 MB
	Notranji pomnilnik	1 GB
	Brezžično omrežje	Intel PRO/Wireless 802.11 a/b/g
Programska oprema	Android verzija	4.1.2
	Različica jedra	2.6.29

Tabela 7.2: Lastnosti navidezne naprave Android.

Strojna oprema	Procesor	Core^{TM}2 Duo 1.6 GHz T5200
	Delovni pomnilnik	4096 MB
	Notranji pomnilnik	WD 320 GB
	Brezžično omrežje	Intel PRO/Wireless 802.11 a/b/g
Programska oprema	Izdaja sistema Windows	Windows 7 Professional, SP1

Tabela 7.3: Lastnosti strežnika spletne storitve.

7.1.2 Navidezna naprava Android

Razvite aplikacije so bile najprej preizkušene na navidezni napravi z imenom *LG_Optimus_4X_HD*. Navidezna naprava se je izvajala na računalniku z enakimi lastnostmi kot strežnik spletne storitve. Lastnosti naprave, ki so pomembne za našo raziskavo, so povzete v Tabeli 7.2.

7.1.3 Strežnik spletne storitve

Na prenosnem računalniku sta se sočasno izvajali obe spletni storitvi. Lastnosti naprave, ki so pomembne za našo raziskavo, so povzete v Tabeli 7.3.

7.2 Razvojno okolje

V tem poglavju bomo predstavili programsko opremo, ki je bila uporabljena pri razvoju spletne storitve in odjemalca. Naprava, ki je bila uporabljena pri

razvoju, ima nameščen JDK (*angl.* Java Development Kit), različica 1.7.

7.2.1 Android SDK

Android SDK (*angl.* Software Development Kit) vključuje celovit nabor razvojnih orodij. Sem spadajo razhroščevalnik, knjižnice, emulator, dokumentacija, vzorčna koda in vodiči. Pri tem diplomskem delu je bil uporabljen Android SDK, različica 22.3.

7.2.2 Upravitelj naprav AVD

Upravitelj naprav AVD (*angl.* Android Virtual Device) je del namestitve Android SDK in omogoča preprosto izdelavo in upravljanje navideznih naprav Android.

7.2.3 Eclipse

Eclipse predstavlja osrednji program pri razvoju spletne storitve in odjemalca. Pomembno je, da že na začetku izberemo pravo različico, ki bo v istem programu združevala možnost izdelave spletnih storitev in mobilnih aplikacij za Android. Pri tem diplomskem delu je bil uporabljen Eclipse *Java EE IDE za spletne razvijalce*, različica Indigo SR2. Dostopen je na naslovu http://www.eclipse.org/downloads/packages/release/indigo/sr2.

Vtičnik razvojnih orodij za Android

Za lažje delo je bil dodatno nameščen vtičnik razvojnih orodij za Android ADT (*angl.* Android Development Tools), ki razširja zmožnosti programa Eclipse s hitro izdelavo aplikacij za Android, uporabniških vmesnikov, možnostjo razhroščevanja in izvoza podpisanih ali nepodpisanih aplikacijskih paketov `.apk`.

DDMS

Strežnik za razhroščevanje in nadzor Dalvik (*angl.* Dalvik Debug Monitor Server) je poseben pogled programa Eclipse, ki se pojavi po namestitvi Android SDK. Omogoča posredovanje vrat, zajem zaslona na napravi, informacije o nitih in kopicah, spremljanje dnevnika, procesov, stanja radia, dohodnih klicev in sporočil SMS, podatkov o lokaciji in mnogo več.

Strežnik spletne storitve

Spletna storitev za svoje delovanje potrebuje strežnik. Ker v naši raziskavi primerjamo dva različna pristopa razvoja spletnih storitev, je za nepristransko analizo zelo pomembno, da se obe spletni storitvi izvajata na istem strežniku. Pri tem diplomskem delu je bil uporabljen strežnik *Apache Tomcat*, različica 7.0.50.

Apache Tomcat je spletni vsebnik, ki omogoča izvajanje *servletov* in spletnih aplikacij, ki temeljijo na JSP (*angl.* JavaServer Pages). *Servleti* so javanski razredi, ki razširjajo strežnikove sposobnosti.

7.2.4 SoapUI

SoapUI je odprtokodno orodje za preizkušanje spletnih storitev, ki omogoča pregled in klice spletnih storitev, simulacije, funkcionalno preizkušanje, obremenitvene in združljivostne preizkuse. Pri tem diplomskem delu je bil uporabljen SoapUI, različica 4.6.4. Dostopen je na naslovu http://www.soapui.org/.

SoapUI je bil uporabljen za preizkušanje spletne storitve SOAP, preden smo to vključili v mobilno aplikacijo Android.

Poglavje 8

Implementacija spletne storitve in odjemalca

Spoznali smo eksperimentalno okolje in programsko opremo, sedaj pa si bomo ogledali spletno storitev in odjemalca za oba obravnavana pristopa. V programu Eclipse smo tako naredili štiri ločene projekte, in sicer spletno storitev SOAP, spletno storitev REST, odjemalca SOAP za Android in odjemalca REST za Android. Razvoj vsakega projekta je potekal vzporedno za SOAP in REST, tako da programska koda uporablja enako logiko za spletno storitev in odjemalca. Enak je tudi uporabniški vmesnik obeh razvitih aplikacij. Razlike se pojavljajo predvsem v uporabljenih knjižnicah in si jih bomo ogledali za vsak projekt posebej.

8.1 Spletna storitev

Spletno storitev sestavljata dve javno dostopni metodi:

- `pridobiDatoteko(int velikost)` in
- `pridobiDatotekoX(int velikost)`.

Prva metoda je namenjena prenosu datotek iz datotečnega sistema strežnika. Pri naši raziskavi smo kot datoteke uporabili slike različnih velikosti. V primeru, da metoda kot velikost prejme število 1, se iz datotečnega sistema

naloži slika 1, v primeru, da kot velikost prejme število 2, se naloži slika 2 in tako naprej. Naložena slika se nato pretvori v niz znakov. Spletna storitev naredi novo instanco razreda `Paket`, ki kot parametra prejme enoznačni identifikator in niz znakov slike, in jo vrne kot odgovor odjemalcu.

Druga metoda je namenjena prenosu datotek, ki fizično ne obstajajo v datotečnem sistemu strežnika, ampak se ustvarijo glede na vrednost parametra `velikost`. V primeru, da metoda kot velikost prejme število 1000, se najprej naredi prazna tabela ustrezne velikosti, ki se nato napolni s poljubnimi znaki. Spletna storitev iz tabele znakov ustvari nov niz znakov, ki ga, podobno kot pri prvi metodi, uporabi za izdelavo nove instance razreda, ki jo vrne kot odgovor odjemalcu. Pri tej metodi smo predpostavili, da en znak zavzame en bajt prostora. Iz tega sledi, da število 1000 pomeni velikost datoteke 1 kB.

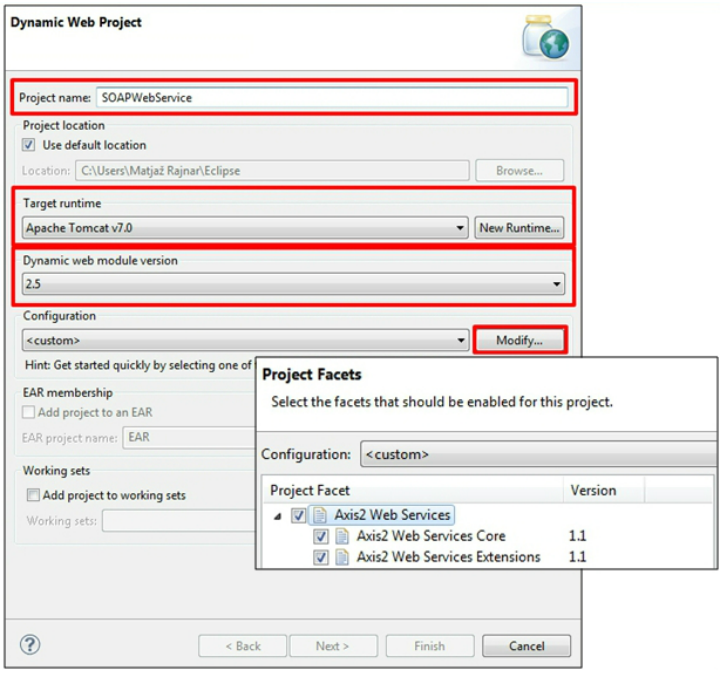
Spoznali smo ogrodje spletne storitve, sedaj pa si bomo ogledali še uporabljene knjižnice in postopek izdelave nove spletne storitve SOAP in REST s programom Eclipse.

8.1.1 Spletna storitev SOAP

Za izdelavo spletne storitve SOAP je bila uporabljena implementacija *Apache Axis2*, različica 1.6.2. *Apache Axis2* spletnim aplikacijam omogoča izgradnjo vmesnikov spletnih storitev, lahko pa deluje tudi kot samostojen strežnik.

Namestitev *Apache Axis2* in vključitev v program Eclipse:

- prenesemo arhiv na naslovu http://apache.spinellicreations.com//axis/axis2/java/core/1.6.2/axis2-1.6.2-bin.zip,
- vsebino arhiva razširimo na `C:\axis2-1.6.2`,
- odpremo program Eclipse in v menijski vrstici izberemo **Window** > **Preferences** > **Web Services** > **Axis2 Preferences**,
- na zavihku **Axis2 Runtime** kliknemo na gumb **Browse**,



Slika 8.1: Priprava projekta za spletno storitev SOAP.

- v pogovornem oknu nastavimo lokacijo iz druge točke in zaključimo namestitev s klikom na gumb **OK**.

Po uspešni vzpostavitvi razvojnega okolja in knjižnic, se lahko lotimo implementacije spletne storitve. Slika 8.1 prikazuje, kako naredimo nov dinamičen spletni projekt za spletno storitev SOAP:

- odpremo program Eclipse in v menijski vrstici izberemo **New** > **Dynamic Web Project**,
- izberemo ime projekta **SOAPWebService**, za izvajalno okolje izberemo **Apache Tomcat v7.0**, za različico spletnega modula izberemo **2.5** in kliknemo na gumb **Modify**,
- v pogovornem oknu pod Project Facets označimo možnost **Axis2 Web Services**,
- zaključimo s klikom na gumb **Finish**.

Ustvarili smo nov dinamičen spletni projekt za spletno storitev SOAP. V drevesni strukturi projekta razširimo mapo **Java Resources** in v mapi **src** ustvarimo dva paketa:

- `com.diploma.soap.objekti` in
- `com.diploma.soap.vmesniki`.

Prvi paket vsebuje javanski razred `Paket` — Seznam 8.1. Njegovi objekti so namenjeni shranjevanju podatkov, ki se prenašajo med spletno storitvijo in odjemalcem. Sestavljen je iz dveh konstruktorjev, `Paket()` in `Paket(long id, String podatki)`, in štirih metod za pridobivanje in nastavljanje vrednosti spremenljivkam.

```
package com.diploma.soap.objekti;
import javax.xml.bind.annotation.XmlRootElement;
public class Paket {
        private long id;
        private String podatki;

        public Paket() {
                id = 0;
                podatki = "";
        }

        public Paket(long id, String podatki) {
                this.id = id;
                this.podatki = podatki;
        }

        public long getId() {
                return id;
        }
        public void setId(long id) {
                this.id = id;
        }
        public String getPodatki() {
                return podatki;
        }
```

```
        public void setPodatki(String podatki) {
                this.podatki = podatki;
        }
}
```

Seznam 8.1: Javanski razred `Paket`.

Drugi paket vsebuje javanski razred `ImplementacijaSpletneStoritve`, ki vsebuje dve javno dostopni metodi. Ta razred pravzaprav predstavlja vmesnik med objektom `Paket` in spletom. Ko bo odjemalec zahteval ta objekt, bo razred obdelal zahtevo in vrnil ustrezen odgovor. Podrobneje si bomo ogledali metodo `pridobiDatotekoX(int velikost)` — Seznam 8.2. Njeno delovanje smo opisali na začetku tega poglavja.

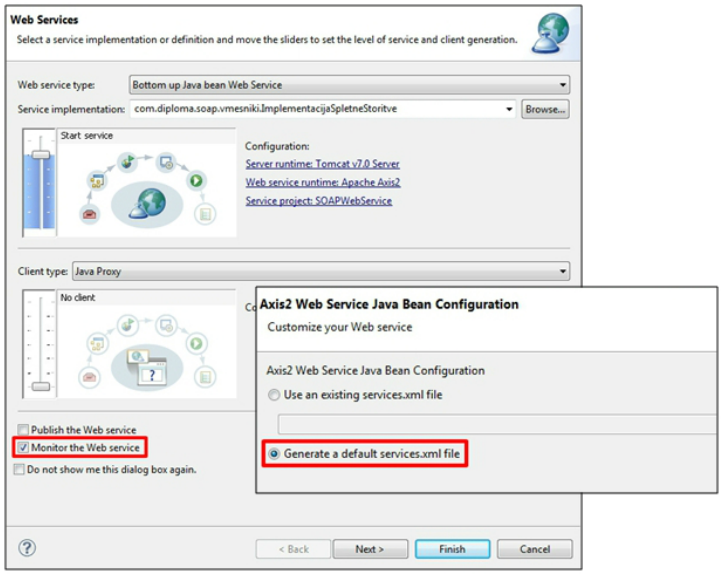
```
public class ImplementacijaSpletneStoritve {
...
        public Paket pridobiDatotekoX(int velikost) {
                char[] znaki = new char[velikost];
                Arrays.fill(znaki, '*');
                String podatki = new String(znaki);
                paket = new Paket(1, podatki);

                return paket;
        }
}
```

Seznam 8.2: Javanski razred `ImplementacijaSpletneStoritve`.

Ko smo napisali vse potrebne metode, je potrebno ustvariti še spletno storitev. Slika 8.2 prikazuje, kako ustvarimo spletno storitev:

- z desnim klikom miške kliknemo na javanski razred `ImplementacijaSpletneStoritve` in izberemo **Web Services** > **Create Web Service**,
- v pogovornem oknu označimo možnost **Monitor the Web service** in kliknemo na gumb **Next**,
- v naslednjem pogovornem oknu pod Axis2 Web Service Java Bean Configuration označimo možnost **Generate a default services.xml file**,



Slika 8.2: Izdelava spletne storitve.

- zaključimo s klikom na gumb **Finish**.

Razvito spletno storitev sedaj lahko zaženemo na strežniku *Tomcat*, tako da z desnim klikom miške kliknemo na projekt **SOAPWebService** in iz menija izberemo **Run As** > **Run on Server**. Dokument WSDL, ki opisuje našo spletno storitev SOAP, se nahaja na naslovu http://192.168.1.5:8080/SOAPWebService/services/ImplementacijaSpletneStoritve?wsdl.

8.1.2 Spletna storitev REST

Za izdelavo spletne storitve REST je bila uporabljena implementacija *Jersey*, različica 1.18. *Java* opredeljuje podporo arhitekturi REST preko specifikacije *JSR 311* (*angl.* Java Specification Request). Ta specifikacija se imenuje *JAX-RS* (*angl.* The Java API for RESTful Web Services) in uporablja anotacije za opredelitev javanskih razredov.

Namestitev *Jersey*:

- prenesemo arhiv na naslovu http://repo1.maven.org/maven2/com/sun/jersey/jersey-archive/1.18/jersey-archive-1.18.zip,
- vsebino arhiva razširimo na `C:\jersey-1.18`.

Vključitev knjižnice v program Eclipse bomo opravili po vzpostavitvi dinamičnega spletnega projekta. Slika 8.3 prikazuje, kako naredimo nov dinamičen spletni projekt za spletno storitev REST:

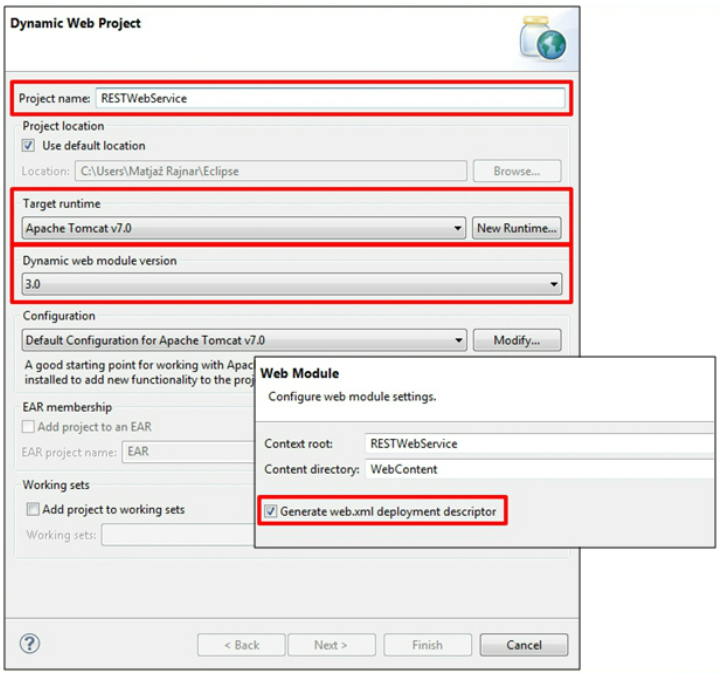
- odpremo program Eclipse in v menijski vrstici izberemo **New** > **Dynamic Web Project**,
- izberemo ime projekta **RESTWebService**, za izvajalno okolje izberemo **Apache Tomcat v7.0**, za različico spletnega modula izberemo **3.0** in kliknemo na gumb **Next**,
- v naslednjem pogovornem oknu označimo možnost **Generate web.xml deployment descriptor**,
- zaključimo s klikom na gumb **Finish**.

Ustvarili smo nov dinamičen spletni projekt za spletno storitev REST. Sedaj moramo k projektu dodati še potrebne datoteke `.jar`, ki smo jih prenesli na začetku tega poglavja:

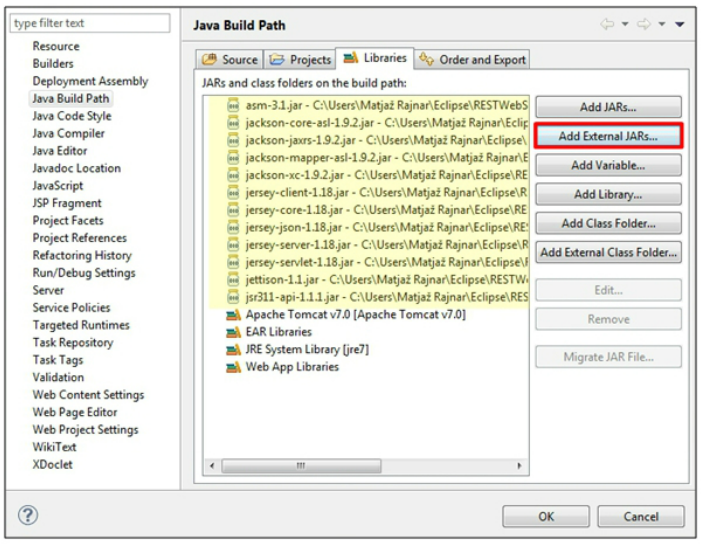
- v drevesni strukturi projekta razširimo mapo **WebContent** in njeno podmapo **WEB-INF**,
- z desnim klikom miške kliknemo na mapo **lib** in izberemo **Import**,
- izberemo mapo, v kateri se nahajajo datoteke `.jar` in jih dodamo k projektu.

Po uvozu datotek je potrebno posodobiti pot za izgradnjo projekta (*angl.* Project Build Path), tako da bo vključevala te datoteke — Slika 8.4:

- z desnim klikom miške kliknemo na projekt **RESTWebService** in iz menija izberemo **Build Path** > **Configure Build Path**,
- na zavihku **Libraries** kliknemo na gumb **Add External JARs** in izberemo mapo, v katero smo uvozili datoteke,
- zaključimo s klikom na gumb **OK**.



Slika 8.3: Priprava projekta za spletno storitev REST.



Slika 8.4: Dodajanje zunanjih knjižnic.

Enako kot pri spletni storitvi SOAP, v drevesni strukturi projekta razširimo mapo **Java Resources** in v mapi **src** ustvarimo dva paketa:

- `com.diploma.rest.objekti` in
- `com.diploma.rest.vmesniki`.

Vsebina prvega paketa je enaka kot pri spletni storitvi SOAP, vsebina drugega paketa pa se, zaradi uporabe anotacij v javanskem razredu `ImplementacijaSpletneStoritve`, nekoliko razlikuje od tistega pri spletni storitvi SOAP. Kljub anotacijam osnovna logika obeh metod ostaja enaka. Podrobneje si bomo ogledali metodo `pridobiDatotekoX(int velikost)` — Seznam 8.3.

```
@Path("/paket")
public class ImplementacijaSpletneStoritve {
...
        @GET
        @Path("/ustvari/{velikost}")
        @Produces(MediaType.APPLICATION_JSON)
        public Paket pridobiDatotekoX(@PathParam("velikost")
           int velikost) {
                char[] znaki = new char[velikost];
                Arrays.fill(znaki, '*');
                String podatki = new String(znaki);
                paket = new Paket(1, podatki);

                return paket;
        }
}
```

Seznam 8.3: Javanski razred `ImplementacijaSpletneStoritve`.

Anotacija `@Path` omogoča določitev identifikatorja URI, ki bo imel dostop do razreda, `@GET` določa, katera metoda bo poklicana, ko bo odjemalec izvedel metodo GET, `@Path`, ki se nahaja znotraj razreda `ImplementacijaSpletneStoritve`, omogoča pripenjanje parametrov identifikatorju URI.

Anotacija `@Path` omogoča tudi uporabo spremenljivk, ki jih navedemo med zavitima oklepajema: `@Path("/ustvari/{velikost}")`. Ko metoda

```
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID"
    version="3.0">
  <display-name>JerseyRESTServer</display-name>
  <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
        <param-name>com.sun.jersey.config.property.packages</param-name>
        <param-value>com.diploma.rest.vmesniki</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Slika 8.5: Vsebina datoteke `web.xml`.

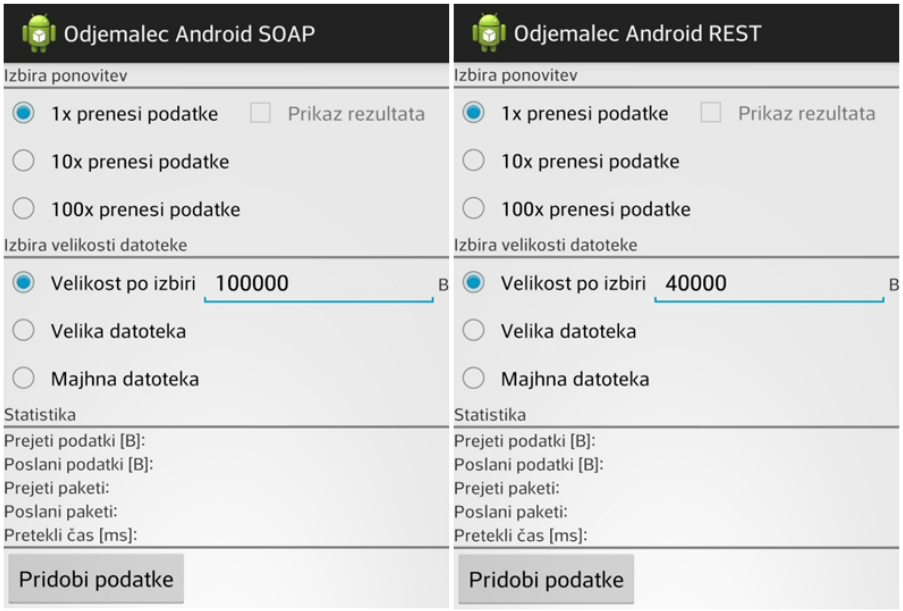
prejme zahtevo z dejansko vrednostjo spremenljivke `velikost`, vrne ustrezen objekt, anotacija `@Produces` pa določi, v kakšni obliki se bo objekt prenesel do odjemalca.

Preden spletno storitev zaženemo na strežniku *Tomcat*, je potrebno urediti datoteko `web.xml`, ker želimo vse zahteve usmeriti k *servletu* `Jersey REST Service`. Prav tako želimo spletni storitvi povedati, kje se nahajajo razredi, ki jih želimo ponuditi našemu odjemalcu:

- v drevesni strukturi projekta razširimo mapo **WebContent** in njeno podmapo **WEB-INF**,
- odpremo datoteko `web.xml` in jo uredimo v skladu s Sliko 8.5.

Dokument XML opredeljuje `Jersey REST Service`, ki se nahaja v razredu `ServletContainer`. Značka <`init-param`> omogoča uporabo razredov, ki se nahajajo v paketu `com.diploma.rest.vmesniki` in tako skrbi za preslikovanje med identifikatorji URI in javansko kodo. Pri znački <`servlet-mapping`> nastavimo značko <`url-pattern`> na `/rest/*`.

Razvito spletno storitev sedaj lahko zaženemo na strežniku *Tomcat*, tako da z desnim klikom miške kliknemo na projekt **RESTWebService** in iz menija izberemo **Run As** > **Run on Server**. Osnovni naslov spletne storitve



Slika 8.6: Uporabniški vmesnik odjemalca SOAP in odjemalca REST.

ima obliko:

- http://192.168.1.5:8080/RESTWebService/rest/

Sestavljen je iz imena projekta in vrednosti, ki smo jo nastavili znački <`url-pattern`>. Za tem sledi ime razreda, v katerem se nahaja implementacija spletne storitve. V našem primeru je to `/paket`. Na koncu sledi še ime metode.

8.2 Odjemalec za Android

V tem poglavju se ponovno vračamo k mobilni platformi Android. Za razvoj odjemalca bomo uporabili knjižnice in pristope, ki smo si jih ogledali na začetku tega diplomskega dela. Slika 8.6 prikazuje uporabniški vmesnik odjemalca SOAP in odjemalca REST za Android.

Uporabniški vmesnik odjemalca je sestavljen iz izbire ponovitev in izbire velikosti datoteke. Izbiramo lahko med enim, desetimi ali stotimi prenosi da-

toteke. Velikost datoteke lahko določimo sami ali pa izberemo veliko oziroma majhno datoteko, ki se nahaja na strežniku spletne storitve. Gre pravzaprav za dve različno veliki sliki, ki jih spletna storitev preko metode ponudi odjemalcu. V primeru, da izberemo en prenos velike ali majhne datoteke, lahko omogočimo tudi možnost za prikaz rezultata. Ko odjemalec prejme potrebne podatke, posebna metoda prikaže sliko.

Na uporabniškem vmesniku si lahko ogledamo tudi statistiko prejetih in poslanih podatkov v bajtih, število prejetih in poslanih paketov in pretekli čas v milisekundah. Razlog za vključitev statistike v mobilno aplikacijo je bolj natančno merjenje podatkov. Programi tretjih oseb namreč podatke merijo na ravni celotne mobilne aplikacije, nas pa zanima samo promet, ki je bil ustvarjen pri pošiljanju zahteve spletni storitvi in obdelavi prejetega odgovora.

Odjemalec je opremljen tudi z osnovnimi varnostnimi mehanizmi in prikazom napredka. Uporabnik lahko podatke od spletne storitve pridobi šele, ko je izbral ustrezno število ponovitev in velikost datoteke. Pri izboru velikosti po izbiri je potrebno vnesti številčno vrednost večjo ali enako nič.

Spoznali smo uporabniški vmesnik odjemalca, sedaj pa si bomo ogledali še uporabljene knjižnice in postopek izdelave novega projekta SOAP in REST za Android s programom Eclipse.

8.2.1 Odjemalec SOAP za Android

Za dostopanje do spletne storitve SOAP z mobilne naprave Android je bila uporabljena knjižnica kSOAP2 Android, različica 3.1.1. Gre za dovršeno, prilagodljivo, odprtokodno in popolnoma brezplačno knjižnico, ki je bila zasnovana posebej za mobilno platformo Android.

Namestitev knjižnice kSOAP2 Android:

- na spletni strani http://code.google.com/p/ksoap2-android/ izberemo zavihek **Source** in v repozitoriju poiščemo datoteko `ksoap2-android-assembly-3.1.1-jar-with-dependencies.jar`,

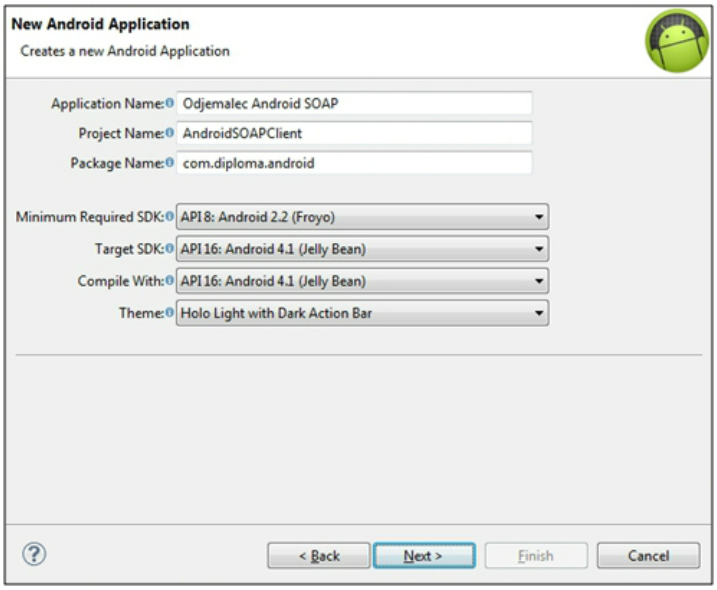
- datoteko shranimo na `C:\ksoap2`.

Vključitev knjižnice v program Eclipse bomo opravili po vzpostavitvi projekta za aplikacijo Android. V nadaljevanju si bomo ogledali, kako naredimo nov projekt za aplikacijo Android, ki bo imel vlogo odjemalca SOAP:

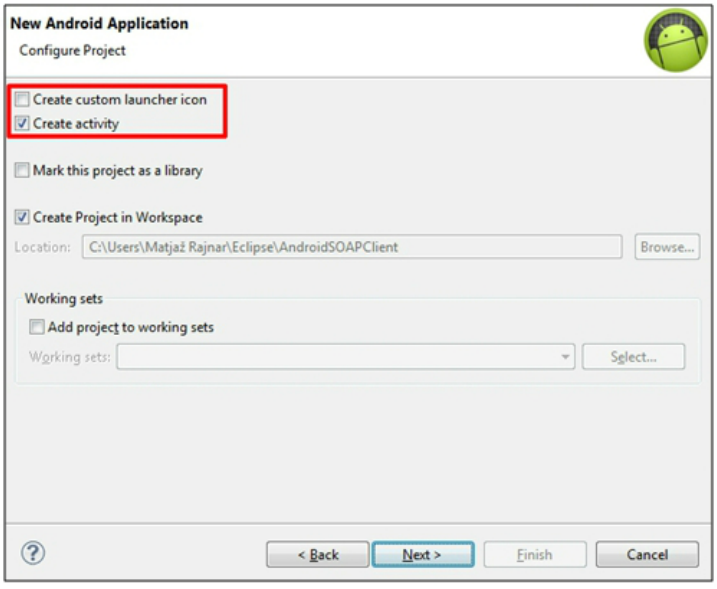
- odpremo program Eclipse in v menijski vrstici izberemo **New** > **Project** > **Android** > **Android Application Project**,
- izberemo ime aplikacije **Odjemalec Android SOAP**, ki bo prikazano uporabnikom, ime projekta **AndroidSOAPClient** in ime paketa `com.diploma.android` — Slika 8.7,
- za minimalen zahtevan SDK pustimo privzeto vrednost,
- za ciljni SDK izberemo **API 16: Android 4.1 (Jelly Bean)**, ker bomo razvito aplikacijo preizkušali na napravi Android, ki ima nameščeno enako različico operacijskega sistema — Slika 8.7,
- za prevajanje prav tako izberemo **API 16: Android 4.1 (Jelly Bean)** in kliknemo na gumb **Next** — Slika 8.7,
- v naslednjem pogovornem oknu odznačimo možnost **Create custom launcher icon** in preverimo, če je označena možnost **Create activity** — Slika 8.8,
- kliknemo na gumb **Next**,
- v naslednjem pogovornem oknu izberemo predlogo **BlankActivity** in zaključimo s klikom na gumb **Finish**.

Ustvarili smo nov projekt za aplikacijo Android, ki bo imel vlogo odjemalca SOAP. Sedaj moramo k projektu dodati še knjižnico kSOAP2 Android, ki smo jo prenesli na začetku tega poglavja:

- v drevesni strukturi projekta z desnim klikom miške kliknemo na mapo **libs** in izberemo **Import**,
- izberemo mapo, v kateri se nahaja knjižnica kSOAP2 Android in jo dodamo k projektu.



Slika 8.7: Priprava projekta za odjemalca SOAP — 1.



Slika 8.8: Priprava projekta za odjemalca SOAP — 2.

Po uvozu knjižnice je potrebno posodobiti pot za izgradnjo projekta, kar smo že opisali v poglavju o spletni storitvi REST. Projekt za aplikacijo Android sestavljajo naslednje pomembnejše datoteke:

1. Javanski razred `AndroidSOAPClient`, ki razširja razred `Activity`. Najprej si bomo ogledali pomembnejše metode in osnovno delovanje mobilne aplikacije:

   - `onCreate` se izvede ob zagonu mobilne aplikacije in pokliče metodo `setContentView`, ki prikaže uporabniški vmesnik mobilne aplikacije. Z metodo `findViewById` pridobimo dostop do elementov uporabniškega vmesnika. S pomočjo razreda `TrafficStats` preverimo tudi, ali naprava podpira analizo omrežnega prometa.
   - `pridobiPodatke` se izvede ob kliku na gumb **Pridobi podatke**. Metoda nastavi začetne vrednosti spremenljivkam in prebere vrednosti, ki jih je uporabnik nastavil na uporabniškem vmesniku. V primeru, da ni napak, metoda naredi novo instanco razreda `WebServiceTask` in jo izvede, sicer pa je uporabnik obveščen o napaki.

   Znotraj razreda `AndroidSOAPClient` se nahaja zasebni razred `WebServiceTask`, ki razširja razred `AsyncTask`. Njegova naloga je povezava na spletno storitev, prenos podatkov in prikaz rezultata. Razred `AsyncTask` omogoča pravilno in enostavno uporabo niti uporabniškega vmesnika (*angl.* UI thread). Namenjen je izvajanju operacij v ozadju in posredovanju rezultatov niti uporabniškega vmesnika. Sestavljen je iz štirih korakov imenovanih `onPreExecute`, `doInBackground`, `onProgressUpdate` in `onPostExecute`.

   V metodi `onPreExecute` si shranimo vrednosti prejetih in poslanih podatkov v bajtih, število prejetih in poslanih paketov in pretekli čas v milisekundah. Na uporabniškem vmesniku prikažemo obvestilo, da poteka pridobivanje podatkov od spletne storitve.

Metoda `doInBackground`, ki je prikazana na Seznamu 8.4, predstavlja osrednji del odjemalca SOAP za Android. Najprej si oglejmo vlogo štirih globalnih spremenljivk:

- `NAMESPACE` hrani vrednost atributa `targetNamespace` iz dokumenta WSDL. V našem primeru ima spremenljivka vrednost http://vmesniki.soap.diploma.com.
- `URL` predstavlja naslov, na katerem je dostopen dokument WSDL. V našem primeru ima spremenljivka vrednost http://192.168.1.5:8080/SOAPWebService/services/ImplementacijaSpletneStoritve?wsdl.
- `METHOD_NAME` predstavlja operacijo spletne storitve in ima v našem primeru lahko dve različni vrednosti: **pridobiDatoteko**, če datoteko prenašamo iz datotečnega sistema strežnika, in **pridobiDatotekoX**, če želimo ustvariti datoteko določene velikosti.
- `SOAP_ACTION` je zloženka spremenljivk `NAMESPACE` in `METHOD_NAME`. Ena izmed možnih vrednosti spremenljivke je lahko http://vmesniki.soap.diploma.com/pridobiDatoteko.

Vrednosti zadnjih dveh spremenljivk sta odvisni od uporabnikove izbire na uporabniškem vmesniku in se nastavita v metodi `pridobiPodatke`.

```
protected String doInBackground(String... urls) {
        String rezultat = "";

        try {
                SoapObject so = new SoapObject(NAMESPACE,
                    METHOD_NAME);

                // povezavi dodamo parametre
                PropertyInfo pi = new PropertyInfo();
                pi.setName("velikost");
                pi.setValue(urls[0]);
                pi.setType(String.class);
                so.addProperty(pi);
```

```
                // ustvarimo ovojnico SOAP
                SoapSerializationEnvelope sse = new
                        SoapSerializationEnvelope(
                           SoapEnvelope.VER11);
                sse.setOutputSoapObject(so);

                androidHttpTransport = new
                   HttpTransportSE(URL);
                androidHttpTransport.call(SOAP_ACTION,
                   sse);

                SoapObject objekt = (SoapObject)sse.
                   getResponse();
                rezultat = objekt.getProperty(1).toString
                   ();

                androidHttpTransport.reset();

        } catch(Exception e) {
                e.printStackTrace();
        }

        return rezultat;
}
```

Seznam 8.4: Metoda `doInBackground` — 1.

Metoda najprej naredi objekt `SoapObject`, ki ga bo potrebovala za izgradnjo zahteve SOAP. Obe metodi spletne storitve za svoje delovanje potrebujeta parameter `velikost`. Njegova vrednost je odvisna od uporabnikove izbire in se nahaja na začetku tabele `urls`. Za dodajanje potrebnih parametrov, ki se bodo poslali skupaj z zahtevo SOAP, metoda naredi nov objekt `PropertyInfo`, nastavi vrednosti spremenljivkam in ga vključi v objekt `SoapObject`. Metoda nato ustvari ovojnico SOAP, spremenljivka `SoapEnvelope.VER11` označuje različico SOAP 1.1, in ji dodeli objekt `SoapObject`. Sedaj je potrebno samo še po-

slati zahtevo SOAP strežniku spletne storitve. Metoda najprej naredi objekt `HttpTransportSE`, ki predstavlja prenosno plast HTTP, nato pa izvede klic z uporabo spremenljivke `SOAP_ACTION` in ovojnice SOAP. Na koncu metoda samo še obdela odgovor spletne storitve z uporabo metode `getResponse`. Prejeti objekt vsebuje enoznačni identifikator, ki se nahaja na ničtem mestu, in podatke, ki se nahajajo na prvem mestu. Metoda prejete podatke shrani v spremenljivko `rezultat` in ponastavi povezavo.

V metodi `onPostExecute` prikažemo rezultat v primeru, da je uporabnik izbral možnost za prikaz rezultata na uporabniškem vmesniku. Ponovno izmerimo vrednosti prejetih in poslanih podatkov v bajtih, število prejetih in poslanih paketov in pretekli čas v milisekundah ter jih odštejemo od začetnih vrednosti. Rezultate prikažemo na uporabniškem vmesniku in ponovimo prenos, če je uporabnik na uporabniškem vmesniku izbral več kot eno ponovitev. Na koncu zapremo obvestilo, da poteka pridobivanje podatkov od strežnika spletne storitve, ki smo ga prikazali v metodi `onPreExecute`.

2. `res\layout\activity_android_soapclient.xml` določa postavitev elementov mobilne aplikacije na uporabniškem vmesniku. Sestavljajo ga naslednji elementi:

   - `Button` predstavlja gumb, ki mu lahko z atributom `android:onClick` dodelimo metodo, ki se sproži ob kliku nanj. Ta mora biti javna in kot edini parameter prejeti pogled `View`. Z njim uporabnik zahteva podatke od spletne storitve.
   - `CheckBox` predstavlja posebno vrsto gumba, ki ima lahko dve stanji. Ko je označen, vrne metoda `isChecked` vrednost resnično, sicer pa neresnično. Z njim uporabnik izbere, ali želi na uporabniškem vmesniku prikazati rezultat.
   - `EditText` je izpeljan iz elementa `TextView`, vendar s to razliko, da omogoča urejanje vsebine. Z njim uporabnik lahko vnese velikost

po izbiri. S pomočjo atributa `android:inputType` lahko vnos omejimo samo na številčne vrednosti.

- `LinearLayout` je postavitev, ki svoje elemente razporedi v en stolpec ali eno vrstico. Kako bodo razporejeni elementi, določa atribut `android:orientation`, ki ima privzeto horizontalno vrednost.
- `RadioButton` predstavlja posebno vrsto gumba, ki ima lahko dve stanji. Ko je označen, vrne metoda `isChecked` vrednost resnično, sicer pa neresnično. Za razliko od gumba `CheckBox`, gumba `RadioButton` uporabnik ne more odznačiti, ko je enkrat označen.
- `RadioGroup` se običajno uporablja za združevanje gumbov `RadioButton` v skupine. Izbira gumba, ki pripada določeni skupini, samodejno odznači vse ostale gumbe iz iste skupine. Kako bodo razporejeni gumbi, določa atribut `android:orientation`, ki ima privzeto horizontalno vrednost. Z njimi uporabnik izbira med ponovitvami in velikostjo datoteke.
- `TextView` se uporablja za prikazovanje besedila na uporabniškem vmesniku mobilne aplikacije. Lahko prikazuje statično vsebino, kjer je vrednost opredeljena v datoteki `strings.xml`, ali pa dinamično vsebino, ki se elementu doda programsko.
- `View` predstavlja osnovni gradnik za komponente uporabniškega vmesnika. Na zaslonu zavzema pravokotno območje in je odgovoren za izris in obravnavo dogodkov. Na uporabniškem vmesniku mobilne aplikacije smo z njim prikazali vodoravne ločilne črte.

Večina zgoraj naštetih elementov ima z atributom `android:id` določen tudi enoznačni identifikator, ki omogoča dostopanje do vrednosti elementov in nastavljanje njihove vrednosti.

3. `res\values\strings.xml` vsebuje vrednosti uporabljenih atributov. Vrednosti lahko določimo tudi neposredno v datoteki XML, ki določa postavitev elementov mobilne aplikacije na uporabniškem vmesniku.

Prednost tega pristopa se pokaže predvsem pri prevajanju mobilne aplikacije v druge jezike, ker imamo vse vrednosti zbrane na enem mestu. Seznam 8.5 prikazuje izsek te datoteke.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
        <string name="izbira_ponovitev">Izbira ponovitev
           </string>
        <string name="izbira_velikosti">Izbira velikosti
           datoteke</string>
        <string name="prikazi">Prikaz rezultata</string>
        <string name="x1">1x prenesi podatke</string>
        <string name="x10">10x prenesi podatke</string>
        <string name="x100">100x prenesi podatke</string>
        ...
</resources>
```

Seznam 8.5: Izsek datoteke `res\values\strings.xml`.

4. `AndroidManifest.xml` vsebuje pomembne podatke mobilne aplikacije kot so ime paketa, minimalen in ciljni SDK, dovoljenja. Da bo naša mobilna aplikacija lahko dostopala do spletne storitve, ji moramo omogočiti povezavo z internetom. Odpremo datoteko `AndroidManifest.xml` in jo uredimo v skladu s Sliko 8.9.

Naša mobilna aplikacija je sedaj pripravljena na prvi zagon. V programu Eclipse z desnim klikom miške kliknemo na projekt **AndroidSOAPClient** in iz menija izberemo **Run As** > **Android Application**. V primeru, da razvito aplikacijo želimo naložiti na mobilno napravo, mora ta biti v načinu **Odpravljanje napak USB**.

8.2.2 Odjemalec REST za Android

Za dostopanje do spletne storitve REST z mobilne naprave Android ne potrebujemo dodatnih knjižnic, ampak uporabimo kar knjižnico Apache HttpClient, ki je del namestitve Android SDK. Knjižnica poenostavi delo z zah-

Slika 8.9: Vsebina datoteke `AndroidManifest.xml`.

tevami HTTP. Za pridobivanje in pošiljanje podatkov se uporablja razred `HttpClient`.

Enako kot v prejšnjem poglavju, naredimo nov projekt za aplikacijo Android, ki bo imel vlogo odjemalca REST. Za ime aplikacije tokrat izberemo **Odjemalec Android REST**, za ime projekta pa **AndroidRESTClient**.

Projekt za aplikacijo Android sestavljajo enake datoteke kot pri odjemalcu SOAP za Android. Podrobneje si bomo ogledali samo metodo `doInBackground`, ki je prikazana na Seznamu 8.6, saj se pri uporabi arhitekture REST precej razlikuje, in dve novi metodi:

- `pridobiParametreHttp` in
- `pretvoriVhodnePodatkeVNiz`.

Tako kot metoda `doInBackground`, se ti dve metodi nahajata znotraj zasebnega razreda `WebServiceTask` v javanskem razredu `AndroidRESTClient`. Ostale metode tega razreda so enake kot pri odjemalcu SOAP za Android in jih ne bomo ponovno obravnavali.

Najprej si oglejmo vlogo treh globalnih spremenljivk:

- `SERVICE_URL` predstavlja naslov, na katerem se nahaja implementacija spletne storitve. V našem primeru ima spremenljivka vrednost

http://192.168.1.5:8080/RESTWebService/rest/paket.

- `CONN_TIMEOUT` je časovna omejitev za povezavo s spletno storitvijo. V našem primeru ima spremenljivka vrednost 3000 milisekund.
- `SOCKET_TIMEOUT` je časovna omejitev za odgovor spletne storitve. V našem primeru ima spremenljivka vrednost 5000 milisekund.

Za dostopanje do določene metode spletne storitve na konec naslova dodamo ime metode. Vrednost naslova je tako odvisna od uporabnikove izbire na uporabniškem vmesniku in se nastavi v metodi `pridobiPodatke`.

```
protected String doInBackground(String... urls) {
        String rezultat = "";
        HttpClient hc = new DefaultHttpClient(
           pridobiParametreHttp());
        HttpResponse hr = null;

        try {
                HttpGet hg = new HttpGet(urls[0]);
                hr = hc.execute(hg);

        } catch(Exception e) {
                e.printStackTrace();
        }

        if (hr != null) {
                try {
                        rezultat = pretvoriVhodnePodatkeVNiz
                                (hr.getEntity().getContent())
                                   ;

                        JSONObject json = new JSONObject(
                           rezultat);
                        rezultat = json.getString("podatki");

                } catch(Exception e) {
                        e.printStackTrace();
                }
```

```
                return rezultat;

        } else {
                return rezultat;
        }
}
```

Seznam 8.6: Metoda `doInBackground` — 2.

Metoda `doInBackground` najprej naredi objekt `DefaultHttpClient`, ki kot parameter prejme metodo `pridobiParametreHttp()` — Seznam 8.7.

```
private HttpParams pridobiParametreHttp() {
        HttpParams hp = new BasicHttpParams();
        HttpConnectionParams.setConnectionTimeout(hp,
           CONN_TIMEOUT);
        HttpConnectionParams.setSoTimeout(hp, SOCKET_TIMEOUT)
           ;

        return hp;
}
```

Seznam 8.7: Metoda `pridobiParametreHttp()`.

Ta nastavi parametre HTTP; časovno omejitev za povezavo s spletno storitvijo na vrednost spremenljivke `CONN_TIMEOUT`, časovno omejitev za odgovor spletne storitve pa na vrednost spremenljivke `SOCKET_TIMEOUT`. Za pridobitev podatkov iz strežnika metoda naredi objekt `HttpGet`. Za razliko od odjemalca SOAP za Android, začetek tabele `urls` vsebuje naslov spletne storitve, metodo, ki se bo izvedla, in vrednost parametra `velikost`. Naslov http://192.168.1.5:8080/RESTWebService/rest/paket/ustvari/100 tako pomeni, da bo spletna storitev ustvarila datoteko velikosti 100 bajtov in jo vrnila odjemalcu. Metoda nato pridobi objekt `HttpResponse` z uporabo metode `execute`, ki kot parameter prejme objekt `HttpGet`. Na koncu metoda samo še obdela odgovor spletne storitve z uporabo metode `pretvoriVhodnePodatkeVNiz` — Seznam 8.8, ki kot parameter prejme vhodni tok iz objekta `HttpResponse`, in rezultat shrani v spremenljivko `rezultat`. Sledi izdelava

novega objekta JSON, ki vsebuje dva para ključ/vrednost:

- `id` predstavlja enoznačni identifikator in
- `podatki` predstavljajo prejeto vsebino.

```
private String pretvoriVhodnePodatkeVNiz(InputStream is) {
        String vrstica = "";
        StringBuilder sb = new StringBuilder();

        // InputStream ovijemo v BufferedReader
        BufferedReader br = new BufferedReader(new
           InputStreamReader(is));

        try {
                // beremo odgovor
                while ((vrstica = br.readLine()) != null) {
                        sb.append(vrstica);
                }

        } catch(Exception e) {
                e.printStackTrace();
        }

        return sb.toString();
}
```

Seznam 8.8: Metoda `pretvoriVhodnePodatkeVNiz`.

Metoda `pretvoriVhodnePodatkeVNiz` je tako namenjena obdelavi podatkov, ki jih je odjemalec REST za Android prejel od spletne storitve. Metoda najprej naredi objekt `StringBuilder`, ki bo hranil prejete podatke. Sledi izdelava objekta `BufferedReader`, ki kot parameter prejme vhodni tok. Metoda nato v zanki bere vrstice iz objekta `BufferedReader` in jih pripenja objektu `StringBuilder`. Na koncu metoda pretvori vsebino objekta `StringBuilder` v niz in ga vrne kot rezultat.

Datoteka `activity_android_restclient.xml` določa postavitev elementov mobilne aplikacije na uporabniškem vmesniku. Edina razlika med obema

uporabniškima vmesnikoma je ime aplikacije, ki je nastavljeno v datoteki `res\values\strings.xml`, ki vsebuje vrednosti uporabljenih atributov. Da bo naša mobilna aplikacija lahko dostopala do spletne storitve, ji moramo omogočiti povezavo z internetom. Postopek smo že opisali pri odjemalcu SOAP za Android.

Naša mobilna aplikacija je sedaj pripravljena na prvi zagon. V programu Eclipse z desnim klikom miške kliknemo na projekt **AndroidRESTClient** in iz menija izberemo **Run As** > **Android Application**. V primeru, da razvito aplikacijo želimo naložiti na mobilno napravo, mora ta biti v načinu **Odpravljanje napak USB**.

Poglavje 9

Rezultati

Za merjenje prenesenih podatkov je bil uporabljen razred `TrafficStats`, ki omogoča analizo omrežnega prometa na vseh omrežnih vmesnikih, na mobilnem vmesniku ali na ravni določene aplikacije z uporabo enoznačnega identifikatorja. Za oba pristopa smo spremljali količino prejetih in poslanih podatkov v bajtih, število prejetih in poslanih paketov in pretekli čas v milisekundah.

Uporabili smo naslednje metode razreda `TrafficStats`:

- `getTotalRxBytes` vrne količino prejetih podatkov v bajtih,
- `getTotalTxBytes` vrne količino poslanih podatkov v bajtih,
- `getTotalRxPackets` vrne število prejetih paketov in
- `getTotalTxPackets` vrne število poslanih paketov.

Za merjenje časa je bila uporabljena metoda `nanoTime` iz razreda `System`. Metoda vrača trenutno vrednost sistemske ure v nanosekundah. Za potrebe meritev smo izmerjen čas pretvorili v milisekunde.

Vse meritve lahko opravimo tudi v programu Eclipse v pogledu DDMS, kjer imamo, poleg prejetih in poslanih podatkov v bajtih ter števila prejetih in poslanih paketov, že izračunano hitrost prenosa, ki nam je razred `TrafficStats` sam po sebi ne vrne. Kljub temu so meritve z njim veliko bolj natančne, ker jih lahko opravimo na poljubnem mestu v programski kodi.

Pogled DDMS namreč podatke meri na ravni celotne mobilne aplikacije in je težko razlikovati med prometom posameznih metod.

Omrežna statistika deluje samo na napravah z različico jedra od 3.0 naprej. Navidezne naprave Android izdelane z upraviteljem naprav AVD v glavnem uporabljajo različico jedra 2.6.29, tako da jih ne moremo uporabljati za opravljanje meritev. Poleg omenjene težave so navidezne naprave tudi zelo počasne.

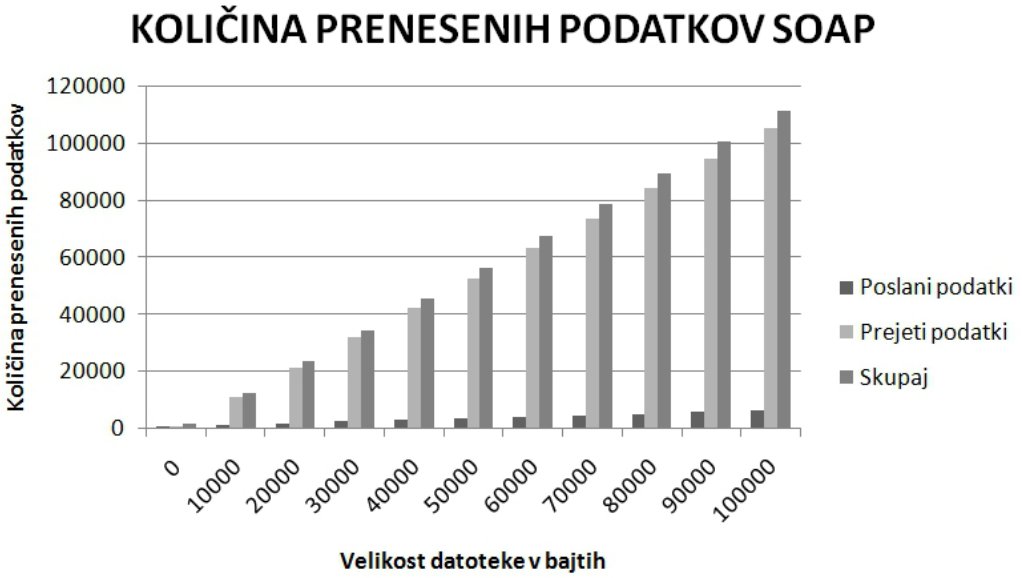
Za meritve je bila uporabljena metoda `pridobiDatotekoX`, ki smo jo podrobno spoznali v poglavju o implementaciji spletne storitve in odjemalca. V nadaljevanju si bomo ogledali rezultate meritev pri prenosu datotek različnih velikosti. Za oba pristopa bomo primerjali količino prenesenih podatkov, število prenesenih paketov, čas prenosa in prepustnost.

9.1 Količina prenesenih podatkov

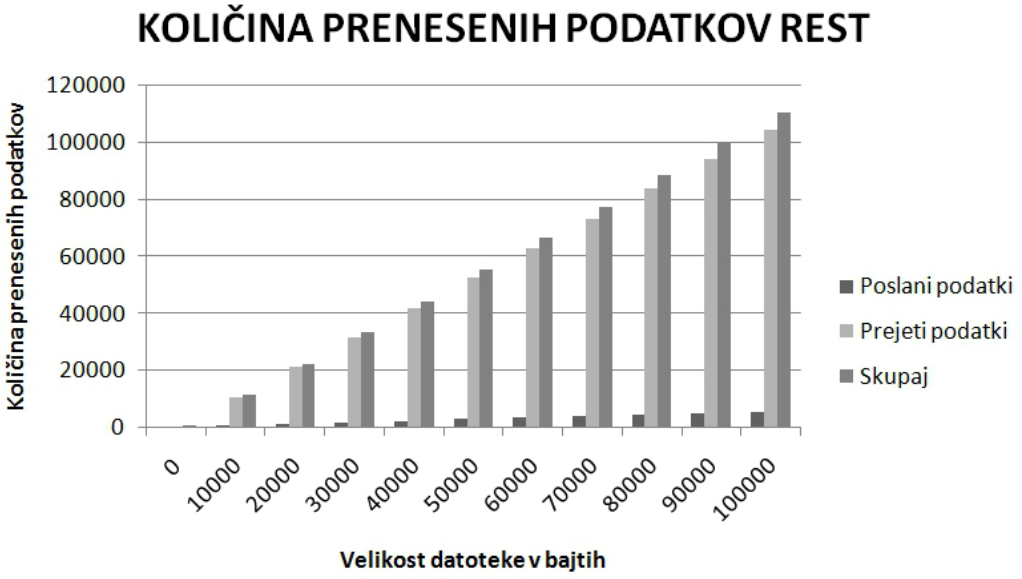
Zanima nas dejanska količina prenesenih podatkov v bajtih. Najprej si bomo ogledali rezultate meritev posebej za SOAP — Slika 9.1 in REST — Slika 9.2, nato pa bomo primerjali rezultate na skupnem grafu — Slika 9.3, na katerem je prikazana vsota poslanih in prejetih podatkov za oba pristopa. Na abscisni osi grafov se nahaja velikost datoteke v bajtih, na ordinatni pa dejanska količina prenesenih podatkov v bajtih. Poslani podatki predstavljajo zahtevo, ki jo odjemalec pošlje strežniku spletne storitve, prejeti podatki pa vsebujejo strežnikov odgovor.

Iz primerjave rezultatov na skupnem grafu — Slika 9.3 je razvidno, da spletna storitev in odjemalec, ki temeljita na protokolu SOAP, za prenos enako velike datoteke zahtevata nekoliko večjo količino podatkov. Rezultati iz grafa namreč prikazujejo prenose večjih datotek, kjer koristna vsebina popolnoma prevlada, zato je količina prenesenih podatkov zelo blizu skupaj. Pri prenosu manjših datotek so razlike med obema pristopoma precej večje.

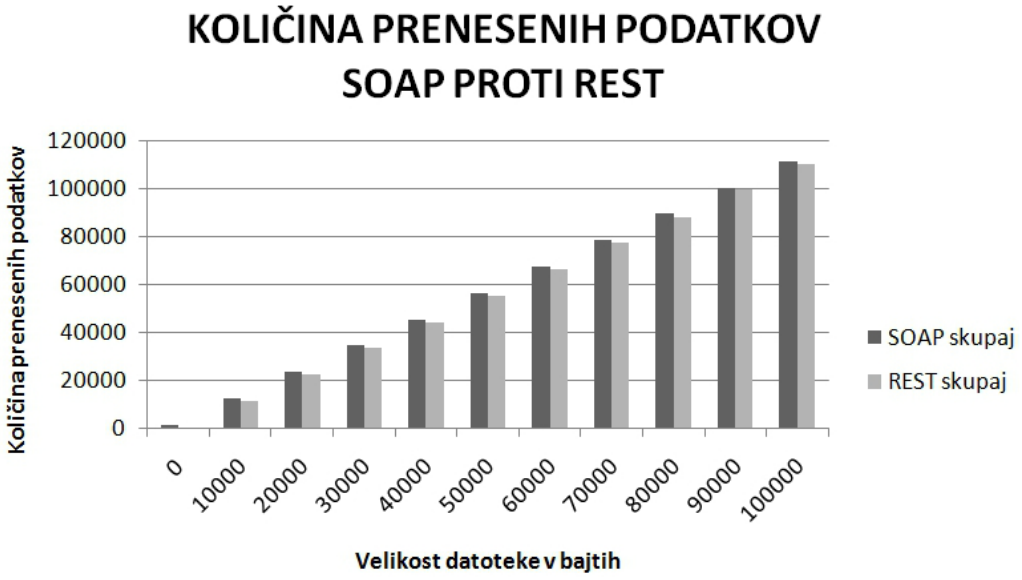
Za boljšo predstavo, kakšna je dejanska razlika v količini prenesenih podatkov, si bomo v Tabeli 9.1 ogledali velikost prazne zahteve in odgovora za



Slika 9.1: Količina prenesenih podatkov SOAP.



Slika 9.2: Količina prenesenih podatkov REST.



Slika 9.3: Količina prenesenih podatkov SOAP proti REST.

	Velikost datoteke [B]	Poslani podatki [B]	Prejeti podatki [B]	Skupaj [B]
SOAP	0	1064	787	**1851**
REST	0	459	341	**800**

Tabela 9.1: Količina prenesenih podatkov prazne zahteve in odgovora za SOAP in REST.

SOAP in REST.

Iz Tabele 9.1 je razvidno, da uporaba pristopa SOAP zahteva dvakrat večjo količino podatkov. Pri mobilnih aplikacijah, ki preko omenjenih pristopov prenašajo manjše količine podatkov, je to velik prihranek. Poleg tega so mobilni podatki v večini primerov plačljivi, zato si prizadevamo, da mobilna aplikacija naredi čim manj prometa.

Slika 9.4: Število prenesenih paketov SOAP proti REST.

9.2 Število prenesenih paketov

Zanima nas število prenesenih paketov. Ogledali si bomo rezultate meritev na skupnem grafu — Slika 9.4, na katerem je prikazana vsota poslanih in prejetih paketov za oba pristopa. Na abscisni osi grafa se nahaja velikost datoteke v kilobajtih, na ordinatni pa število prenesenih paketov.

Iz primerjave rezultatov na skupnem grafu — Slika 9.4 je razvidno, da je število prenesenih paketov pri uporabi obeh pristopov v večini primerov popolnoma enako. To je tudi pričakovan rezultat, saj so bile datoteke, ki smo jih prenašali, enake velikosti.

Za boljšo predstavo, kakšna je dejanska razlika v številu poslanih in prejetih paketov, si bomo v Tabeli 9.2 ogledali prazno zahtevo in odgovor za SOAP in REST.

Razlika med poslanimi in prejetimi paketi je vedno enaka dva. To je posledica tri-smernega rokovanja (*angl.* three-way handshake) pri vzpostavljanju in prekinjanju povezave, ki ga pogosto označujemo kot SYN, SYN-ACK,

	Velikost datoteke [B]	Poslani paketi	Prejeti paketi	Skupaj
SOAP	0	5	3	**8**
REST	0	5	3	**8**

Tabela 9.2: Število poslanih in prejetih paketov prazne zahteve in odgovora za SOAP in REST.

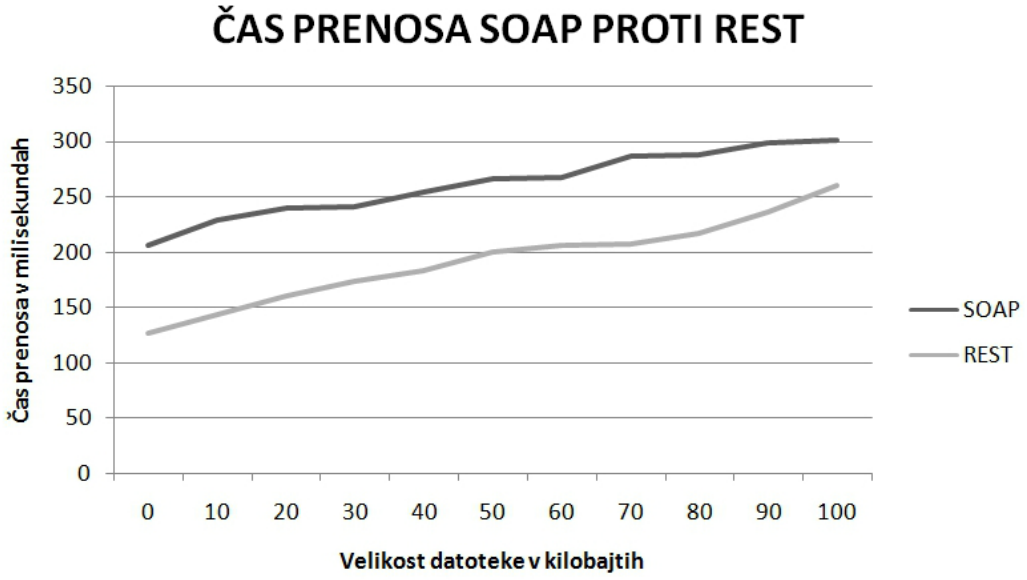
ACK, ker se med odjemalcem in strežnikom spletne storitve prenesejo trije paketi:

1. odjemalec strežniku spletne storitve pošlje zahtevo SYN za vzpostavitev ali prekinitev povezave,
2. strežnik spletne storitve odjemalcu odgovori, tako da pošlje svojo zahtevo SYN in z ACK potrdi odjemalčevo zahtevo SYN,
3. odjemalec z ACK potrdi strežnikovo zahtevo SYN.

9.3 Čas prenosa

Zanima nas čas, ki preteče od trenutka, ko odjemalec zahteva datoteko določene velikosti, do trenutka, ko je datoteka v celoti prenesena iz strežnika spletne storitve. Ogledali si bomo rezultate meritev na skupnem grafu — Slika 9.5, na katerem je prikazan čas prenosa za oba pristopa. Za vsako velikost datoteke je bil izračunan povprečen čas prenosa desetih meritev. Na abscisni osi grafa se nahaja velikost datoteke v kilobajtih, na ordinatni pa čas prenosa v milisekundah.

Iz primerjave rezultatov na skupnem grafu — Slika 9.5 je razvidno, da spletna storitev in odjemalec, ki temeljita na protokolu SOAP, za prenos enako velike datoteke zahtevata opazno več časa. Z večanjem velikosti datoteke naraščajo tudi časi prenosa, razlika med obema pristopoma pa ostaja v povprečju enaka.

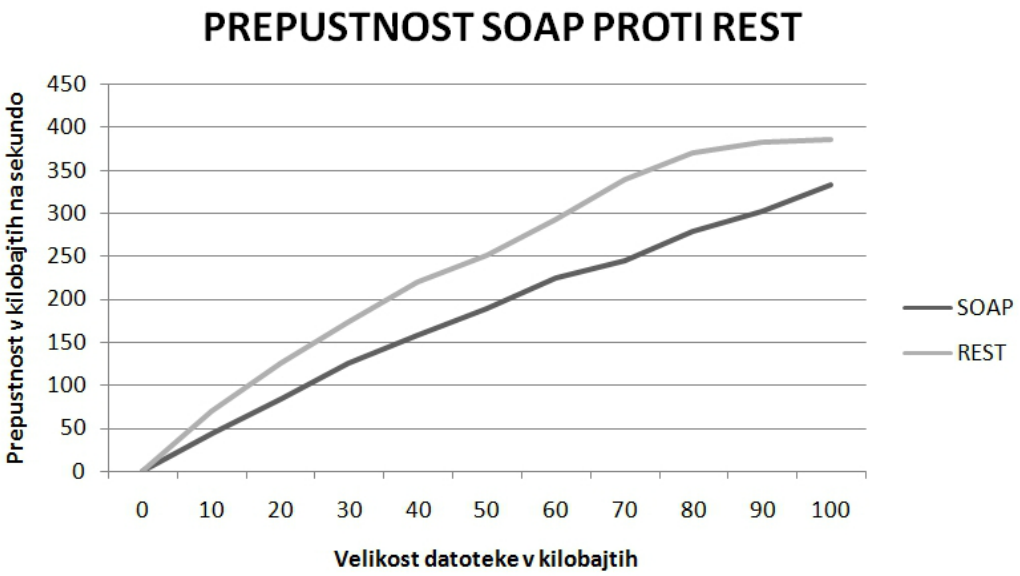


Slika 9.5: Čas prenosa SOAP proti REST.

9.4 Prepustnost

Zanima nas prepustnost v kilobajtih na sekundo. Prepustnost je definirana kot povprečna stopnja uspešnega prenosa podatkov preko kanala. Izračunamo jo tako, da velikost datoteke delimo s časom prenosa. Ogledali si bomo rezultate meritev na skupnem grafu — Slika 9.6, na katerem je prikazana prepustnost za oba pristopa. Na abscisni osi grafa se nahaja velikost datoteke v kilobajtih, na ordinatni pa prepustnost v kilobajtih na sekundo.

Zaradi večjega časa prenosa pri uporabi protokola SOAP je pričakovano manjša tudi njegova prepustnost, kar se lepo vidi na grafu — Slika 9.6. Opazimo lahko, da se prepustnost pri protokolu SOAP z velikostjo datoteke spreminja zelo malo. Nekoliko drugače pa je pri arhitekturi REST, kjer se od 80 kilobajtov naprej prepustnost glede na velikost datoteke zmanjša, vendar je še vedno večja kot pri protokolu SOAP. Spletne storitve namreč niso namenjene prenašanju velikih datotek in postanejo neučinkovite predvsem zaradi serializacije in različnih vrst kodiranja. Pravilen pristop za prenašanje veli-



Slika 9.6: Prepustnost SOAP proti REST.

kih datotek je uporaba ustreznih protokolov, SOAP in REST pa se uporabita samo za pridobivanje naslova URL, na katerem se nahaja datoteka.

Poglavje 10

Sklepne ugotovitve

Spoznali smo pojem spletnih storitev in si ogledali dva vodilna pristopa pri njihovem razvoju za mobilne naprave Android. Kljub temu da Android ne zagotavlja podpore za neposredno klicanje spletnih storitev SOAP, lahko s pomočjo ročne gradnje sporočil SOAP in obstoječih knjižnic za podporo SOAP izdelamo odjemalce za mobilne aplikacije. Za razliko od spletnih storitev SOAP, za dostopanje do spletnih storitev REST ne potrebujemo dodatnih knjižnic, ampak uporabimo kar knjižnico Android HttpClient, ki je razvijalcem na voljo že od zgodnjih različic mobilne platforme Android. Velikost mobilnih aplikacij je zaradi tega tudi ustrezno manjša, kar je še posebej pomembno pri napravah z omejenimi viri.

Razumevanje arhitekture REST je lažje, če poznamo protokol HTTP. REST zahteva več vloženega napora pri razvoju strežnika spletne storitve, medtem ko je razvoj odjemalcev razmeroma preprost. Na drugi strani je razvoj strežnika spletne storitve SOAP zaradi večje podpore in razširjenosti veliko bolj preprost, imamo pa številne težave pri razvoju odjemalcev, še posebej zaradi slabe podpore na mobilni platformi Android. Tudi sami smo naleteli na težave pri izbiri prave implementacije. Pri uporabi implementacije *GlassFish Metro* in njene komponente *JAX-WS RI*, nam na mobilni aplikaciji nikakor ni uspelo pridobiti odgovora spletne storitve, čeprav je ta bila preizkušena s programom SoapUI. Zato smo se na koncu odločili za

implementacijo *Apache Axis2*, ki je enostavnejša za uporabo in ima veliko boljšo podporo. Implementacija spletne storitve in odjemalca REST je bila izvedena veliko hitreje in brez težav.

Primerjava rezultatov meritev je pokazala, da se pri pošiljanju prazne zahteve in odgovora SOAP prenese dvakrat večja količina podatkov, kar je posledica besedne zasnove XML, dokumenta WSDL in številnih drugih dejavnikov. Ker so mobilni podatki v večini primerov plačljivi, si prizadevamo, da mobilna aplikacija naredi čim manj prometa. Meritve časa prenosa in prepustnosti pri prenosu enako velike datoteke so prav tako bile v prid arhitekturi REST. Mobilne aplikacije, ki temeljijo na tej arhitekturi so tako opazno hitrejše in v enakem času lahko prenesejo večjo količino podatkov. Zanimivo bi si bilo ogledati tudi primerjavo obeh pristopov pri večjem številu naprav. Ker smo pri izvedbi meritev razpolagali zgolj z eno mobilno napravo Android, takšne primerjave nismo mogli opraviti.

Kateri pristop pri razvoju spletnih storitev bomo uporabili, je temeljna odločitev, ki lahko vpliva na uspeh mobilne aplikacije. Tako SOAP kot tudi REST sta primerna za implementacijo podobne funkcionalnosti. V splošnem bi naj SOAP uporabljali, kadar potrebujemo posebno funkcijo, ki je na voljo samo pri protokolu SOAP, sicer pa je uporaba arhitekture REST najboljša možnost. Dobro implementirane spletne storitve REST ponujajo najboljše razmerje med produktivnostjo razvijalcev, izkoristkom pasovne širine, hitrostjo, prepustnostjo in robustnostjo. Vse to prispeva k uspehu in pravočasnemu razvoju mobilne aplikacije.

Na podlagi prednosti in slabosti posameznega pristopa ter rezultatov meritev lahko zaključimo, da je REST bolj razširljiv, interoperabilen in ponuja boljše rezultate, medtem ko se SOAP bolje odreže pri aplikacijah, kjer je poudarek na varnosti in zanesljivosti.

Literatura

[1] G. Alonso. *Web Services: Concepts, Architectures and Applications.* Springer, 2004, pogl. 5.

[2] (2014) Web services. Dostopno na: http://www.webopedia.com/TERM/W/Web_Services.html

[3] (2014) The Proliferation of Mobile Devices. Dostopno na: http://www.ni.com/white-paper/13788/en/

[4] (2014) Introduction to Web Services. Dostopno na: http://www.w3schools.com/webservices/ws_intro.asp

[5] (2014) SOAP. Dostopno na: http://en.wikipedia.org/wiki/SOAP

[6] (2014) SOAP Introduction. Dostopno na: http://www.w3schools.com/webservices/ws_soap_intro.asp

[7] (2014) The structure of a SOAP message. Dostopno na: http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp

[8] (2014) Web Services Description Language. Dostopno na: http://www.w3.org/TR/wsdl

[9] (2014) WSDL Documents. Dostopno na: http://www.w3schools.com/webservices/ws_wsdl_documents.asp

[10] J. Cox (2014) SOAP vs. REST For Mobile Services. Dostopno na: https://www.captechconsulting.com/blog/jack-cox/soap-vs-rest-mobile-services

[11] L. Richardson, S. Ruby. *RESTful Web Services*. O'ReillyMedia, 2007.

[12] L. Jordan, P. Greyling. *Practical Android Projects*. Apress, 2011, pogl. 7.

[13] D. Crockford (2014) The application/json Media Type for JavaScript Object Notation. Dostopno na: http://www.ietf.org/rfc/rfc4627.txt

[14] S. Kirk (2014) Speaking SOAP with Android. Dostopno na: http://www.shanekirk.com/2011/11/speaking-soap-with-android/

[15] S. Komatineni, D. MacLean. *Pro Android 4*. Apress, 2012, pogl. 15.

[16] J. McHugh (2014) Low Bandwidth SOAP. Dostopno na: http://www.xml.com/pub/a/ws/2003/08/19/ksoap.html

[17] S. Oh. *Web Service Architecture for Mobile Computing*. ProQuest, 2006.

[18] N. Apte, K. Deutsch, R. Jain. "Wireless SOAP: Optimizations for Mobile Wireless Web Services", v zborniku: DoCoMo USA Labs. Chiba, Japan, maj 2005, str. 1178–1179.

[19] T. Kozel, A. Slaby. "Mobile access into information systems", v zborniku: Int. Conf. on Information Technology Interfaces. 2008, str. 851–856.

[20] V. Dobjanschi. "Developing Android REST client applications", v zborniku: Google I/O. Moscone Center, San Francisco, maj 2010, str. 851–856.